STADIA
HELSINGIN AMMATTIKORKEAKOULU

TEKNIIKAN JA LIIKENTEEN TOIMIALA

Tietotekniikka

Ohjelmistotekniikka

INSINÖÖRITYÖ

RASKAAN KALUSTON TIEDONKERUUJÄRJESTELMÄ SULAUTETTUNA JAVA-OHJELMISTONA

Työn tekijä: Asmo Salonen
Työn valvoja: Simo Silander
Työn ohjaaja: Jouni Heikkinen

Työ hyväksytty: __. __. 2008

Simo Silander
lehtori

STADIA
HELSINGIN AMMATTIKORKEAKOULU

ALKULAUSE

Tämä insinöörityö tehtiin syksyn 2007 ja kevään 2008 aikana Sesca Logistics IT Oy:lle osana Tekesin VAMOS-hanketta. Kiitän kaikkia VAMOS-hankkeessa tutuksi tulleita eri yritysten edustajia mahdollisuudesta olla mukana mielenkiintoisessa projektissa, Sesca Logistics IT Oy:n ohjelmistokehitystiimiä järjestelmän toteutuksen avustamisesta sekä Aplicom Oy:n teknistä tukea annetusta avusta erinäisissä A1 Flex -ajoneuvotietokoneen ohjelmistoon liittyvissä ongelmatilanteissa.

Erityiset kiitokset osoitan Mikko Toivoselle (projektipäällikkö/Gli Logistics Oy) VAMOS-hankkeen vetämisestä sekä Jouni Heikkiselle (toimialajohtaja/Sesca Logistics IT Oy) mahdollisuudesta tehdä insinöörityö.

Helsingissä 20.4.2008

Asmo Salonen

STADIA
HELSINGIN AMMATTIKORKEAKOULU

INSINÖÖRITYÖN TIIVISTELMÄ

Tekijä: Asmo Salonen	
Työn nimi: Raskaan kaluston tiedonkeruujärjestelmä sulautettuna Java-ohjelmistona	
Päivämäärä: 20.4.2008	Sivumäärä: 55 s. + 5 liitettä
Koulutusohjelma: Tietotekniikka	Suuntautumisvaihtoehto: Ohjelmistotekniikka
Työn valvoja: lehtori Simo Silander Työn ohjaaja: toimialajohtaja Jouni Heikkinen	

Tämä insinöörityö tehtiin Sesca Logistics IT Oy:lle osana Tekesin VAMOS-hanketta. Logistiikan alan yritysten resursseja on kuluttanut puutteellinen ja hidas tietojen kulku toimiston ja ajoneuvon välillä. Perinteisesti ajoneuvosta tarvittujen tietojen kerääminen on tapahtunut kynällä paperille, ja tiedonkulku on perustunut puhelinyhteyteen toimiston ja ajoneuvon kuljettajan välillä. Työn tavoitteena oli toteuttaa automaattinen tietojenkeruu ajoneuvosta sekä ajoneuvon sisältämistä laitteista. Reaaliaikainen raskaan kaluston tiedonkeruujärjestelmä mahdollistaa ajoneuvoista automaattisesti kerättyjen tietojen helpon seurannan Internetin välityksellä.

Työssä suunniteltiin raskaan kaluston ajoneuvotietokoneen sulautettu ohjelmisto sekä palvelinpään ohjelmisto ajoneuvosta lähetettyjen tietojen vastaanottamiseen. Työ aloitettiin arkkitehtuurisuunnittelulla, jonka tuloksena hahmotettiin järjestelmän olellisimmat komponentit sekä valittiin sopivimmat protokollat komponenttien väliseen tiedonsiirtoon Internet-verkossa. Ajoneuvotietokoneen ohjelmisto toteutettiin sulautettuna ohjelmistona Java ME -teknologialla, jossa hyödynnettiin UML-mallinnusta sekä tunnettuja suunnittelumalleja (design patterns). Palvelinpään tietojen vastaanotto-ohjelmisto toteutettiin Java SE -teknologialla.

Työn lopputuloksena syntyi reaaliaikainen ja hajautettu tietojärjestelmä logistiikan alan tarpeisiin. Tietojärjestelmä kykenee tuottamaan reaalaikaista tietoa ajoneuvon sijainnista sekä ajoneuvon CAN-väylän sisältämistä tiedoista. Lisäksi tietojärjestelmä mahdollistaa tienhoitoajoneuvojen aurojen ja suolasirottimen seurannan. Tietojärjestelmän avulla yritykset voivat seurata kustannustehokkuuttaan ja saada nopeammin yksityiskohtaisempia tietoja ajoneuvojen tilasta.

Avainsanat: Java ME, sulautettu Java-ohjelmisto, raskaan kaluston tiedonkeruujärjestelmä

STADIA
HELSINGIN AMMATTIKORKEAKOULU

ABSTRACT

Name: Asmo Salonen

Title: Implementation of Commercial Vehicle Data Acquisition System Using Embedded Java Software

Date: 20 April 2008 Number of pages: 55

Department: Information Technology Study Programme: Software Engineering

Instructor: Simo Silander, Senior Lecturer

Supervisor: Jouni Heikkinen, Vice President

This final year project was carried out for Sesca Logistics IT Ltd as part of a Tekes project called VAMOS. The resources of the logistics companies have been faced with problems concerning deficient and slow information exchange between the office and the vehicles. Traditionally, the information from the vehicles has been recorded using pen and paper and information exchange has been based on mobile phone connections between the office people and the vehicle drivers. The objective of this final year project was to implement an automatic data acquisition system to collect information produced by the vehicles and vehicle devices. The real time commercial vehicle data acquisition system makes it easy to follow the latest vehicle information via Internet.

One of the main aims of this study was to design and implement the embedded software for the commercial vehicle computer. Another goal was to create the server side software component to receive the information sent from the vehicles. The project was started with architecture design in order to determine the components of the system and also to select the proper protocols for data communications. The vehicle computer software was implemented as embedded software with Java ME technology. The embedded software exploited UML modeling and well-known design patterns. The server side software component was implemented with Java SE technology.

This final year project was successful in creating a real time and embedded information system to be used by companies in the logistics business. The information system is able to produce real time information of vehicle position and CAN bus of the vehicle. In addition, the information system makes it possible to monitor the vehicles' ploughs and salt sprinkler usage. This information system helps the companies to follow their costs and cost efficiency. The vehicle status information is now available for the companies in more detail and much faster, as well.

Keywords: Java ME, Java embedded software, commercial vehicle data acquisition system

SISÄLLYS

ALKULAUSE

TIIVISTELMÄ

ABSTRACT

SISÄLLYS

1 JOHDANTO

Tämän työn tavoitteena on kehittää uusinta teknologiaa hyödyntävä sulautettu Java -ohjelmisto välittämään tietoja langattomasti raskaan kaluston ajoneuvoista. Eteen tulevat haasteet kohdataan ajoneuvotietokoneen Java-olioteknologiaa hyödyntävillä mallinnusmenetelmillä (UML) yhdistettynä tunnetuimpiin oliosuunnittelumalleihin (design patterns). Tällainen lähestymistapa sulautetun ohjelmiston toteuttamiseksi on uusi ja erilainen verrattuna perinteisiin proseduraalisilla kielillä toteutettuihin sulautettuihin ohjelmistoratkaisuihin. Oliopohjainen Java-teknologia takaa monipuoliset ja avoimet ohjelmistokehitystyökalut sulautetun sovelluksen kehittämiseen.

Sesca Logistics IT Oy on mukana Tekesin liiketoiminnan mobiileja ratkaisuja tukevassa VAMOS-hankkeessa GLi Logistics Oy:n, YIT Rakennus Oy:n, Kuljetusliike Suvinen Oy:n sekä Oy Sisu Auto Ab:n kanssa. VAMOS - Liiketoiminnan mobiilit ratkaisut -ohjelma keskittyy langattoman teknologian hyödyntämiseen yritysten liiketoiminnassa monilla eri toimialoilla kuten teollisuudessa, liikenteessä, rakentamisessa ja palveluissa. Tavoitteena on parantaa yritysten tuottavuutta kehittämällä mobiilia teknologiaa hyödyntäviä ratkaisuja liiketoimintojen tehostamiseen. VAMOS tukee mobiiliratkaisujen kehittämistä kaupallisesti kannattaviksi vientikelpoisiksi tuotteiksi. Tavoitteena on kestävä ja kasvava *"Mobile Enterprise"* – liiketoiminta, joka laajentuessaan synnyttää myös uusia korkeatasoisia työpaikkoja. Ohjelmaan kuuluvilla tutkimushankkeilla kehitetään ja ylläpidetään alan osaamista ja luodaan edellytyksiä kaupallisille menestystarinoille. [1.]

Aplicom Oy on kehittänyt modulaarisen ja innovatiivisen sulautetun A1 Flex -ajoneuvotietokoneen monenlaisten langattomien ja M2M-järjestelmien toteuttamiseen. Yleiskäyttöinen ajoneuvotietokone sisältää helpon ja nopean rajapinnan erilaisten sovellusten rakentamiseksi. Sesca Logistics IT Oy:n suunnittelee ja toteuttaa YIT Rakennus Oy:n ja Kuljetusliike Suvinen Oy:n tarpeita vastaavan tiedonkeruujärjestelmän raskaan kaluston ajoneuvoihin soveltaen Aplicomin sulautettua tietokonetta. GLi Logistics Oy:n rooli on olla projektissa hankkeen vetäjä ja raportoija. Sisu Auto tarjoaa kehitettyä järjestelmää asiakkailleen ja suorittaa laitteistoasennukset jo ajoneuvon kokoonpanovaiheessa asiakkaan ostaessa järjestelmän.

Tämä insinöörityö sisältää oliomaailman suunnittelumenetelmiin pohjautuvan ajoneuvotietokoneen ohjelmiston suunnittelun ja toteutuksen. Tämän lisäksi insinöörityö sisältää palvelimelle suunnitellun ja ohjelmoidun oliopohjaisen sovelluksen tietojen vastaanottamiseen ja varastoimiseen tiedonhallintajärjestelmään. Työn ulkopuolelle jäävät erilliset palvelimella olevat raportit, joista asiakas varsinaisesti näkee ajoneuvotietokoneen lähettämät tiedot. Raportteja käsitellään lähinnä asiakkaan saaman hyödykkeen näkökulmasta, kun puolestaan ajoneuvotietokoneen ohjelmistoa ja palvelimella olevaa tietojen vastaanottosovellusta käsitellään myös teknisestä näkökulmasta.

2 TAVOITTEET JA HAASTEET KEHITYSPROJEKTIN YRITYKSISSÄ

Kehitysprojektin tavoitteena on kehittää tiedonkeruujärjestelmä keräämään informaatiota raskaan kaluston ajoneuvoista. Kerättävä informaatio pitää sisällään esimerkiksi ajoneuvon sijaintitiedot, ajoneuvon moottorin tuottamat tiedot, tienhoitolisälaitteiden tuottamat tiedot sekä kuljettajien työajan seurannan. Informaatio esitetään yksinkertaisesti ja havainnollisesti erilaisin tiivistetyin raportein niin, että se hyödyttää asiakasta jokapäiväisissä päätöksenteko-prosesseissa. Kerätty informaatio on saatavissa helposti Internetin välityksellä 24 tuntia vuorokaudessa paikasta riippumatta. Asiakkaan ei tarvitse asentaa erillisiä ohjelmistoja saadakseen informaatiota esille. Osa kehitysprojektin haasteista on asiakkaille yhteisiä ja osa asiakaskohtaisia. Esimerkiksi YIT Rakennus Oy:n tienhoitoajoneuvoista kerätään monenlaista tienhoitoon liittyvää informaatiota kuten maanteiden auraus- ja suolausosuuksia.

Asiakaslähtöinen prosessien läpinäkyvyys, kulujen seurattavuus ja resurssien hallinta ovat kilpailutekijöitä, joihin kuljetusalan yritysten on tulevaisuudessa panostettava. Tämä edellyttää aikaisempaa teknologian hyödyntämistä. Toimintakulttuurin kehittyminen tulee vaatimaan myös yrityksen sisäisiltä ja ulkoisilta tietovirroilta merkittävää tehostumista ja ennen kaikkea seurattavuutta sekä raportoinnin selkeyttämistä yrityksen päätöksenteon tueksi. Erityisesti johdon ja esimiesten tarvitsema informaation laatu ja reaaliaikaisuus sekä verkottuminen ovat merkittäviä. [2.]

Logistiikan kasvava strateginen merkitys on muuttanut myös mainittujen toimialojen toimintaympäristöjä. Yritysten palveluntarjoajien on vaadittaessa pystyttävä tuottamaan kansainvälisiä ratkaisuja joko itsenäisesti tai osana laajempaa palveluverkostoa ja

tietoliikenneverkkoa. Kyseessä olevien toimialojen tulisi vastata haasteisiin kehittämällä omien prosessiensa läpinäkyvyyttä sidosryhmille ja lisätä seurattavuutta toiminnassaan. [2.]

2.1 Teiden kunnossapitopalvelut - YIT Rakennus Oy

Tienhoidon yrityksissä on harvoin hyödynnetty tienhoitolaitteiden tuottamaa informaatiota. Alan yritysten tavoitteena on kerätä yhä tarkemmin tienhoitoajoneuvojen lisälaitteiden tuottama informaatio. Talteen kerättyä tietoa voidaan jälkeenpäin hyödyntää monilla tavoin esimerkiksi suunniteltaessa tienhoitolaitteiden ennakoivaa kunnossapitoa. Automaattinen tienhoitolaitteiden tiedonkeruun hyöty on merkittävä, sillä ajoneuvon suorittamat työvaiheet tulee myös dokumentoitua, ja laadunseuranta on mahdollista toteuttaa tarkemmin. Kerätyt tiedot ovat aina saatavilla, mikä mahdollistaa historiatietojen hakemisen. Auraustoimintaa harjoittavat yritykset voivat seurata reaaliajassa tienhoitoajoneuvojen työn etenemistä ja tarvittaessa reagoida entistä nopeammin auraamattomien tieosuuksien kunnostamiseen. [2.]

Kuvan 1 tienhoitoajoneuvosta kerätään etu- ja sivuauran sekä alusterän asentotietoja, joihin liitetään paikka-, aika- ja auton nopeustieto. Tällöin saadaan kattava tilannekuva auratusta tieosuudesta. Kerätystä tiedosta voidaan jalostaa myös muunlaista tietoa esimerkiksi laskemalla aurojen rasitusta suhteutettuna ajokilometreihin. Tämän tiedon pohjalta pystytään ennakoimaan mahdollisia vikatilanteita ja osataan varautua ajoissa huoltotoimenpiteisiin. Kuluvat osat voidaan vaihtaa ajoissa uusiin niin, että ne eivät hajotessaan aiheuta turhia kaluston poissaolemisia ja viimekädessä lisäkustannuksia. Kuvan 1 tienhoitoajoneuvosta kerätään auratietojen lisäksi suolasirottimelta pariakymmentä erilaista seurattavaa parametriä. Tärkein suolasirottimen seurattavista parametreista on tielle levitetyn suolan määrä.

Kuva 1. Tienhoitoajoneuvo, jossa sivuaura, alusterä ja suolasirotin [2]

Tienhoitolisälaitteiden käyttötietojen keräämisen lisäksi tavoitteena on langaton tietojen siirto yrityksen toimistoon päätöksentekijöiden hyödynnettäväksi. Langaton tiedonsiirto nopeuttaa yrityksen päätöksentekoprosesseja ja palvelee paremmin asiakkaita. Ei tarvitse odottaa työtehtävästä palaavaa ajoneuvoa, kun tietoja saadaan reaaliaikaisesti kustakin ajoneuvon suorittamasta työvaiheesta. Tavoitteena on reaaliaikainen tienhoidon toteutumisen seuraaminen sekä hyvä valmius vastata mahdollisiin asiakaskyselyihin. [2.]

Kolmas tienhoitoalan yritysten tavoite on tarjota sisäisille ja ulkoisille asiakkaille tienhoidon toteutumisen seurantamahdollisuus reaaliaikaisesti esimerkiksi annetusta Internet-osoitteesta. Tällaisia asiakkaita ovat mm. Tiehallinto ja yksittäiset tienkäyttäjät. [2.]

Teiden kunnossapidon haasteena Suomen leveysasteilla on erityisesti keliohjattavuuden puute. Tienhoitoyrityksillä on vaikeuksia ohjata omaa nopeasti muuttuvaa toimintaympäristöään tehokkaasti. Sään vaihtelut ovat yllättäviä eikä tietoa saada siirrettyä ajoneuvon ja konttorin välillä tehokkaasti. Tulisi tietää tarkoin, milloin on suolattu mikäkin tieosuus ja kuinka paljon suolaa tieosuudella käytettiin. Kun levitetyt suolamäärät nähdään reaaliajassa havainnollisesti tieosuuksittain kartan päällä, pystytään johtamaan yhä paremmin tienhoitokaluston liikettä oikeaan suuntaan. Nykyisin keliohjaus perustuu lähinnä työnjohdon luonnollisiin havaintoihin sekä tiesääasemien raportteihin. [2.]

Toinen tienhoitoyritysten haaste on liittää tienhoitolaitteet palvelusovellukseen, jolla saataisiin kerätyksi liiketoiminnan kannalta tärkeää informaatiota. Tienhoitolaitteista ei myöskään saada tietoa esimerkiksi käytetystä maa- ja kiviaineesta. Nykyisten lukuisten eri tietojärjestelmien ylläpito vaatii investointeja, eivätkä tarjotut palvelut vastaa kunnossapidon tarpeita. Teiden kunnossapitoyritykset etsivät sovellusta, jolla saadaan integroiduksi yrityksen tarpeet yhdeksi kokonaisuudeksi. Sovelluksella voidaan ohjata liiketoimintaa, seurata toteutumia sekä tienhoitolaitteiden toimintaa. [2.]

2.2 Maansiirtokuljetukset - Kuljetusliike Suvinen Oy

Maansiirtokuljetusalan yrityksissä ensisijaisena tavoitteena on parantaa tiedonsiirron laatua ja nopeutta yrityksen prosesseissa. Käytännössä se tarkoittaa entistä aikaisempaa reagointia asiakkaiden tekemiin tilauksiin. Yrityksillä on halu lisätä teknologian käyttöä liiketoiminta-prosesseissa, jotta saadaan vähennettyä manuaalisia työvaiheita. Yritysten kilpailuetu voi olla esimerkiksi asiakkaan liiketoimintaa tukevan reaaliaikaisen informaation tarjoaminen kuljetustapahtumista.

Kuva 2. Maansiirtokuljetuksissa käytettävä sora-auto

Kuvassa 2 näkyy maansiirtokuljetuksissa käytettävä sora-auto. Maansiirtologistiikan suurimmaksi haasteeksi on noussut tiedonsiirron hitaus. Tilaus-toimitusketjun tiedonsiirron laatu on heikkoa ja perustuu pitkälti käsinkirjoitettuihin kuormalistoihin sekä tiedon siirtoon puhelimitse konttorin ja ajoneuvon välillä. Asiakkaiden kyselyihin vastaaminen esimerkiksi toimitusajan suhteen on kankeaa ja ainoastaan kuljettajan antaman tiedon varassa. Asiakkailla ei ole mahdollisuutta seurata omaa kuljetusta, ja muutokset kuljetustilauksiin ovat erittäin kankeasti muokattavissa, koska tiedon siirto prosessissa on hidasta. Reagointi asioihin on yleisesti hidasta. Myös ennakointi ja suunnittelu ovat miltei mahdottomia suorittaa. Tieto kulkee hitaasti toimistolle, jolle jää 6-24 tuntia aikaa toteuttaa sovittu toimitus. Manuaalinen työ on runsasta. Samoja tietoja kirjataan lukuisia kertoja eri dokumentteihin ennen kuin toteutunut kuljetus on laskutettu. Työaikaa kuluu runsaasti, mikä vaikuttaa merkittävästi kuljetusyrittäjän tulokseen ja kannattavuuteen. [2.]

Toinen haaste kuljetusliikkeillä on ajoneuvo, jota ei ole onnistuttu yhdistämään yritysten muihin tietojärjestelmiin. Tämä tosiasia hidastaa ja vaikeuttaa kuljetusliikkeiden toimintaa, kun ei ole reaaliaikaista tietoa kuljetettavan tavaran tilasta tai kuljetuksen etenemisestä. Kuljetusyritykset toimivat irrallaan kuljetusten tilaajien ja loppuasiakkaiden tietojärjestelmistä. Kuljetusyritysten omat olemassa olevat tietojärjestelmät eivät ole riittävän avoimia, eikä rajapintoja ole avattu muiden sidosryhmien suuntaan. [2.]

Kuljetusyrityksen johtamisen työkalut ovat alkeellisia. Yrityksillä on hyvin rajalliset mahdollisuudet analysoida omaa kustannustehokkuuttaan, koska työaikaa ja polttoainetta sekä muita kuluja ei kirjata reaaliaikaisesti ja tiedon laatu on heikkoa ja vaikeasti luettavissa. Kuljetusliikkeiden kustannuksista 60 % syntyy työaika- ja polttoainekuluista. Maansiirtokuljetuksista puuttuu myös työkalu kuormakohtaisen tuoton laskemiselle sekä kapasiteettitarpeen arvioinnille. Kuljetusyrittäjät joutuvat arvioimaan ilman kustannusrakennetietoja sekä kulujaan että kapasiteettitarvettaan tarjotessaan palveluja asiakkaille. [2.]

2.3 Ajoneuvovalmistaja - Oy Sisu Auto Ab

Ajoneuvojen valmistajan tavoitteena on kerätä ja etävalvoa ajoneuvon huoltotietoja langattomasti ajoneuvoista asiakkaan ja oman huoltotoiminnan tarpeisiin ja seurantaan. Kuljetusyritykselle on taloudellisempaa huoltaa autot ennen vikojen kasvamista suhteettoman isoiksi. [2.]

Ajoneuvonvalmistajan nykytilan haasteita ovat muunmuassa seuraavat:

1) Ajoneuvosta tulisi pystyä lähettämään tietoa langattomasti reaaliajassa.

2) Ajoneuvon huoltotarve pitäisi pystyä selvittämään etävalvontana.

3) Ajoneuvon käytön ja rasituksen tarkastelu pitäisi pystyä tehdä etävalvontana.

4) Ajoneuvoon tulisi luoda tietotekninen rajapinta, johon muut asiakkaan sidosryhmät voivat vaivatta liittyä.

5) Ajoneuvon valmistajalla pitäisi olla mahdollisuus tarjota ajoneuvoon tai työkoneeseen tehtaalla integroitu ratkaisu, joka tarjoaa asiakkaalle tietohallinnollista lisäarvoa.

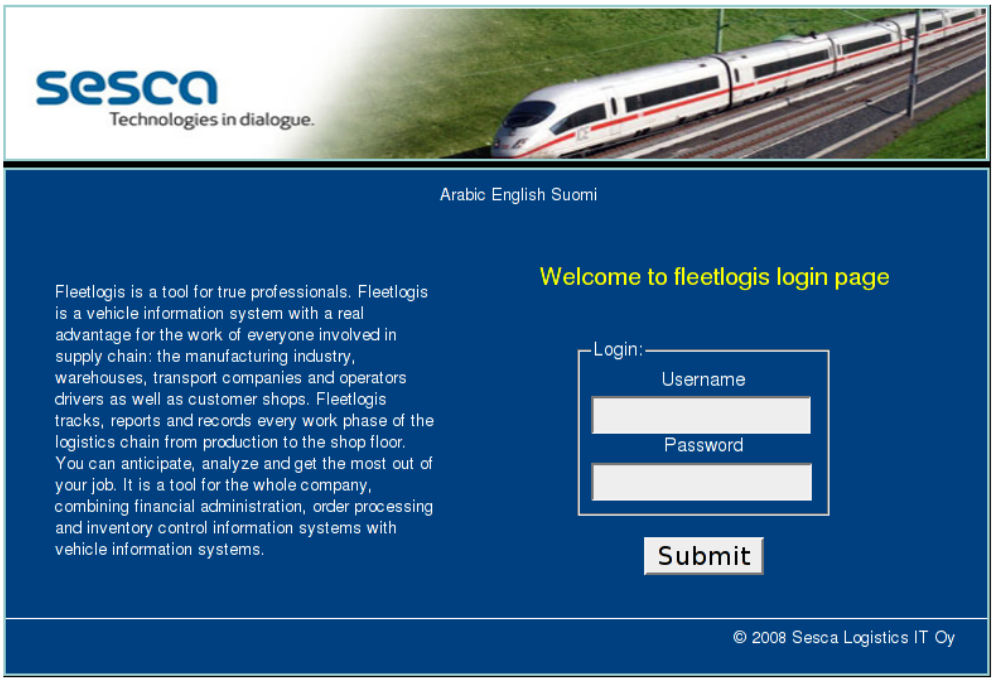
6) Tietoteknisesti tulisi mahdollistaa työkoneen lisäkomponenttien ja lisävarusteiden, kuten tienhoitolaitteiden integrointi. [2.]

3 HAJAUTETTU FLEETLOGIS-TIETOJÄRJESTELMÄ

Fleetlogis on VAMOS-hankkeessa Sesca Logistics IT Oy:n kehittämä kuljetuslogistiikan tietojärjestelmä, jonka asiakkaina ovat ensivaiheessa tienhoito- ja maanrakennusalan yritykset. Näillä aloilla on todettu olevan käytössä jonkin verran tietojärjestelmiä, mutta järjestelmät eivät tuota kuljetustuotannon ja tienhoidon ohjauksen tarvitsemaa tietoa. Fleetlogis on hajautettu ja reaaliaikainen tietojärjestelmä, joka kerää ja välittää itsenäisesti raskaan kaluston ajoneuvojen tilatietoja langattomasti palvelimelle. Fleetlogisilla asiakas löytää oleelliset tiedot isostakin tietomäärästä.

Fleetlogis-tietojärjestelmä poistaa perinteisin menetelmin "kynällä paperille" tehdyt merkinnät asiakkaan prosesseista ja muuttaa nämä tietokoneilla automatisoiduiksi toiminnoiksi. Fleetlogis kerää reaaliaikaisesti tienhoitolisälaitteista saatavan tiedon ja siirtää sen

langattomasti palvelimelle. Fleetlogis tarjoaa asiakkaalle vaivattomasti historiatietojen jäjitettävyyden erilaisista ajoneuvojen tekemistä työsuoritteista. Fleetlogis on yleiskäyttöinen tietojärjestelmä, jonka sisältämät modulit voidaan konfiguroinnilla liittää asiakaskohtaisesti hauluttujen tietojen seurantaan. Fleetlogisin yleiskäytettävyys näkyy sen kykynä palvella monen eri kuljetuslogistiikka-alan tarpeita. Sitä on sovellettu Suomessa tienhoito- ja maansiirtologistiikkaan sekä Kuwaitissa nosturiajoneuvojen seurantaan.



Kuva 3. Fleetlogis-tietojärjestelmän kirjautumissivu

Kuvassa 3 on Fleetlogis-tietojärjestelmän kirjautumissivu. Kirjauduttuaan palveluun asiakas pääsee hakemaan kerättyjä tietoja informatiivisina raportteina. Palvelusovellus mahdollistaa nopean tuotteiden, palvelujen ja resurssien jäljitettävyyden toimitusketjussa hyödyntäen langatonta teknologiaa. Sovelluksen avulla on mahdollista paikantaa yrityksen liikkuvat yksiköt pohjautuen GPS-satelliittipaikannukseen. Se mahdollistaa esimerkiksi ajettujen reittien raportoinnin sekä ajoneuvojen liikkumisen seurannan reaaliaikaisesti paikkaan katsomatta. [2.]

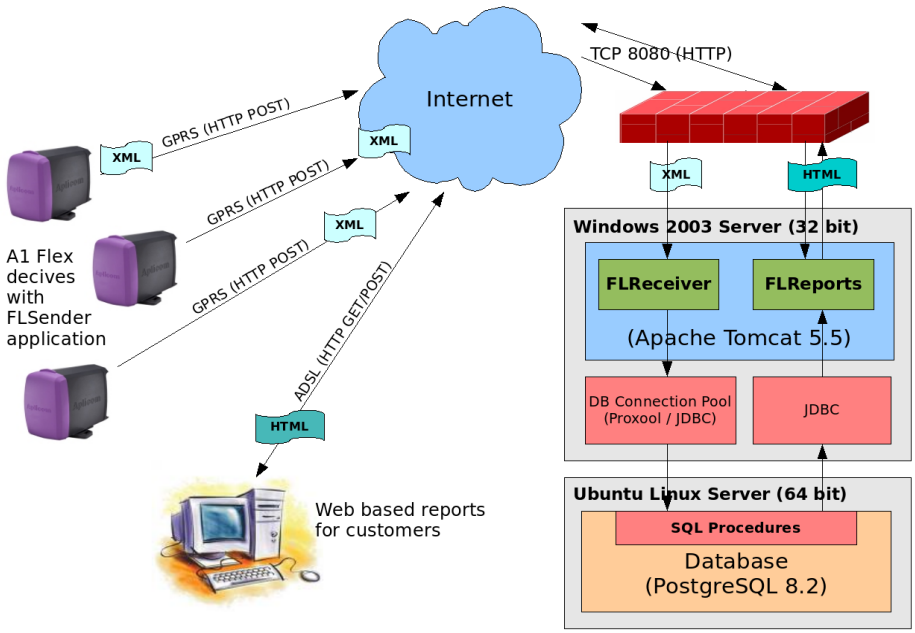
Fleetlogisiin siirretyt raskaan kaluston tiedot antavat asiakkaalle mahdollisuuden reagoida aikaisemmin ja ennakoida tulevia kustannuksia. Yritysten toimintaprosessit voidaan suunnitella entistäkin tarkemmin. Langatonta teknologiaa hyödyntävä Fleetlogis antaa näköalapaikan kustannustehokkaaseen toimintaan. Järjestelmän avulla saatua tietoa pystytään hyödyntämään optimaalisen kapasiteetin määrittämisessä sekä yrityksen resurssien hallinnassa. Sovellus luo hyvän pohjan myös yrityksen kannattavuuslaskelmiin, kun pystytään paremmin kanavoimaan yrityksen menot. [2.]

3.1 Arkkitehtuuri

Kuva 4 havainnollistaa hajautetun Fleetlogis-tietojärjestelmän arkkitehtuuria. Hajautettu Fleetlogis-tietojärjestelmä koostuu kolmesta erillisestä ohjelmistokomponentista:

1) A1 Flex -ajoneuvotietokoneen ohjelmistosta FLSender
2) palvelimella olevasta tietojen vastaanottosovelluksesta FLReceiver
3) palvelimella olevasta ja WWW-selaimella käytettävästä raporttisovelluksesta FLReports.

Komponentit kommunikoivat asiakas-palvelin-arkkitehtuurin mukaisesti. Ajoneuvotietokoneen ohjelmiston (FLSender) ja palvelimella olevan tietojen vastaanottosovelluksen (FLReceiver) välinen kommunikointi perustuu XML-metakieleen. Ajoneuvotietokone lähettää langatonta (GPRS) yhteyttä käyttäen sovellustason protokollalla (HTTP) XML-viestin palvelimelle. HTTP-protokolla käyttää kuljetustason protokollaa (TCP/IP) tietojen siirtämiseen. TCP/IP-protokolla varmistaa automaattisesti pakettien perillemenon. Tietojen vastaanottosovellus tallentaa lähetetyn XML-sanoman tiedot tietokantaan. Asiakas näkee ajoneuvon tiedot WWW-selainpohjaisen käyttöliittymän (FLReports) välityksellä.



Kuva 4. Arkkitehtuurikaavio hajautetusta Fleetlogis-tietojärjestelmästä

Kuvasta 4 nähdään palvelimen tietojärjestelmän olevan hajautettu erilliseen sovelluspalvelimeen ja tietokantapalvelimeen. Tämä ratkaisu lisää tietoturvaa, sillä ajoneuvotietokoneet tai asiakkaan tekemät raporttipyynnöt eivät koskaan ole yhteydessä suoraan tietokantapalvelimeen. Ajoneuvotietokoneen ja asiakkaan kommunikointi tietokantapalvelimeen tapahtuvat sovelluspalvelimen kautta. Tietokantapalvelimelle ohjelmoitujen tallennettujen proseduurien tehtävä on palvella sovelluspalvelimen komponentteja hoitamalla kaikki tietokantaan kohdistuvat pyynnöt ja operaatiot. Sovelluspalvelimen komponentit kutsuvat tietokantapalvelimelta tallennettuja proseduureja tallentaakseen ajoneuvotietokoneen lähettämiä tietoja tai näyttääkseen asiakkaan haluamaa tietoa raportteina. Tallennetut proseduurit edistävät järjestelmän ylläpitoa, koska ne sijaitsevat keskitetysti yhdellä palvelimella ja mahdolliset muutokset tarvitsee tehdä vain yhteen paikkaan.

Sovelluspalvelimen käyttöjärjestelmänä on 32-bittinen Windows 2003 Web Edition, jonka päällä ajetaan Apache Tomcat 5.5 Servlet -palvelinohjelmistoa. Apache Tomcat 5.5 Servlet -palvelinohjelmisto on FLReceiver- ja FLReports-komponenttien suoritusalusta. Tietokantapalvelimen käyttöjärjestelmänä on 64-bittinen versio Ubuntu Linux Server -käyttöjärjestelmästä. Tietokantapalvelimen ohjelmistona käytetään avoimeen lähdekoodiin perustuvaa PostgreSQL 8.2 -tiedonhallintajärjestelmää.

3.2 Saavutettava hyöty

Ajoneuvoissa oleva tietokone kerää automaattisesti monenlaista tietoa. Tieto siirretään lähes reaaliajassa palvelimelle. Palvelimella olevat raportit koostavat kerätyn tiedon havainnolliseksi ja tekevät tietojen selaamisesta yksinkertaista. Kerätyt tiedot ovat tarkasteltavissa WWW-pohjaisen käyttöliittymän avulla. Asiakas saa nopeasti ja helposti tarvitsemansa tiedon nähtäville tukemaan liiketoimintaratkaisujaan.

Tärkeimpiä autosta kerättäviä tietoja on paikkatieto, jolla ajoneuvo voidaan paikantaa. Kuljetusalalla sijaintitieto on arvokasta, koska sitä hyödyntämällä kuljetusjärjestelijät voivat tehdä entistä nopeammin ja tarkemmin päätöksiä kuljetusmääräyksen lähettämisestä oikealle ajoneuvolle. Rahaa säästyy, kun turhia kilometrejä jää ajamatta.

Paikkatieto liitetään lähes kaikkeen muuhun tietoon, jota ajoneuvosta kerätään. Paikkatietoon liitetään erilaiset ajoneuvon suorittamat tehtävät. Kun paikkatietoa kerätään lisäksi tietyn määräajan välein, saadaan piirrettyä yhtenäistä reittiä ajoneuvon suorittamasta tehtävästä.

Paikkatietoa kerätään esimerkiksi aurojen liikkuessa ylös ja alas, jolloin voidaan piirtää kartan päälle auton auraamat tieosuudet.

Ajoneuvon tilannekuva muodostuu mm. ajoneuvossa olevan sähköisen CAN-väylän tiedoista. Lyhenne CAN tulee sanoista Controller Area Network. CAN-väylä on häiriöalttiisiin olosuhteisiin tarkoitettu sarjamuotoinen tiedonsiirtoväylä, jolta saadaan esimerkiksi moottorin tilatiedot. CAN-väylältä kerätään määräajan välein tietoja. Näin saadaan kattavaa kuvaa ajoneuvon toiminnasta. Kun tiedot liitetään paikkatietoon, nähdään kartalta ajoneuvon sijainti, kuka on ajamassa ajoneuvoa ja millä nopeudella auto liikkuu. Sähköiseltä CAN-väylältä kerätään mm.

- ajoneuvon nopeus
- ajoneuvon kuluttama kokonaispolttoainemäärä
- ajoneuvon kulkemat kokonaiskilometrit
- moottorin hetkelliskierrosnopeus
- moottorin kokonaiskäyttötunnit
- kuljettajatunnistus
- työaikakirjaukset.

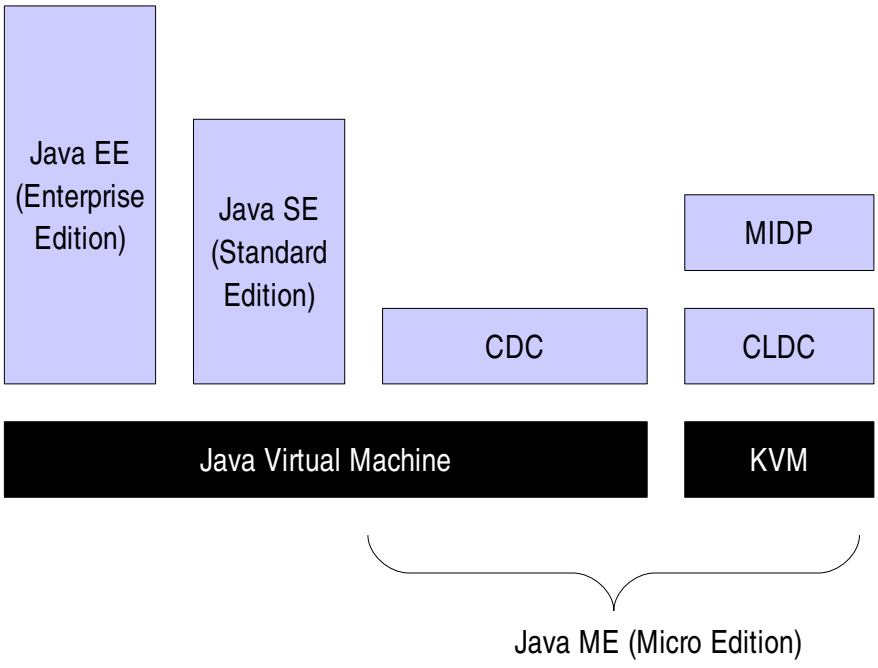
Osa tiedoista tarjoaa pitkän ajan kuluessa arvokasta tietoa ajoneuvon käytöstä. Kerättyjä tietoja voidaan hyödyntää monin tavoin ja lopulta säästetään kustannuksissa. On mahdollista suunnitella esimerkiksi kullekin ajoneuvolle huollot pohjautuen ajettuihin kokonaiskilometreihin ja moottorin kokonaiskäyttötunteihin. Kun määritetty raja-arvo ajoneuvon huollolle lähestyy, järjestelmä ilmoittaa siitä hyvissä ajoin. Näin tiedetään, mitkä ajoneuvot täytyy milloinkin huoltaa ja huoltoajankohdat pystytään määrittämään aikaisemmin. Koska tieto välittyy automaattisesti ajoneuvoista toimistoon, ei enää tarvitse kiertää määräajoin jalkaisin ajoneuvoja lävitse.

Tienhoitoalan yrityksiä kiinnostaa seurata aurojen aurausmatkaa eli montako kilometriä aurat ovat olleet alhalla valitun ajanjakson aikana. Tällä tiedolla tienhoitoalan yritykset pystyvät ennakoitumaan aurojen tarvitsemaan huoltoon ja parempaan käyttövarmuuteen. Yrityksiä kiinnostaa lisäksi teille levitetyn suolan määrä. Tienhoitoyritykset joutuvat maksamaan sakkoja mahdollisesta lisäsuolan ostosta. Mitä tarkemmin yritykset osaavat arvioida tarvitsemansa vuosittaisen suolamäärän sen parempi. Yritykset voivat hyödyntää vuosittaisessa suolamäärän arvioinnissa aiempina vuosina järjestelmän keräämiä suolamääräärien tietoja.

4 SULAUTETTUJEN JÄRJESTELMIEN OHJELMOINTI JAVALLA

4.1 Javan polku sulautettujen järjestelmien toteteutuskieleksi

Java-kielen kehitys alkoi Sun Microsystemsillä osana toista projektia, jonka tarkoituksena oli kehittää edistynyt ohjelmisto sulautettujen järjestelmien verkottamiseen. Ohjelmiston hallintaan tarvittiin pieneen, luotettavaan, siirrettävään, hajautettuun ja reaaliaikaiseen käyttöympäristöön sopiva ohjelmointikieli. Projektin sivutuotteena syntyi täysin uusi ohjelmointikieli Oak, joka sai vaikutteita C++:sta ja Smalltalkista. Kielen pääkehittäjä James Gosling antoi uudelle ohjelmointikielelle työhuoneen ulkopuolella kasvaneen tammen englanninkielisen nimen Oak. Ensimmäinen versio Oakista julkaistiin vuonna 1991, ja myöhemmin kieli sai nykyisin paremmin tunnetun nimen Java. [3.]



Kuva 5. Java 2:ssa ympäristöt on jaoteltu käyttötarkoitusten mukaan [4]

Vuonna 1999 julkaistetun Java 2:n myötä Java-ympäristöt eriytyivät kuvassa 5 esitetyn jaotelman mukaan erilaisiin käyttötarpeisiin. Standardiympäristö Java SE on suunniteltu työpöytäsovellusten tarpeisiin, Java EE on suunniteltu palvelimille ohjelmoitavien sovellusten alustaksi ja Java ME on suunniteltu laitteisiin, joissa on rajoitettu muisti, näyttö tai laskentateho. Java ME kehittyi varteenotettavaksi sulautettujen järjestelmien toteutuskieleksi vuonna 2001, jolloin Nokia, Siemens ja Motorola ilmoittivat Java-tuesta omissa matkapuhelimissaan. Java ME -kirjasto on monipuolinen, ja se sisältää mm. mobiililaitteiden ohjelmointiin tarvittavat rajapinnat. [3; 4.]

4.2 Java ME (Micro Edition)

4.2.1 Konfiguraatio

Konfiguraatio on määritelmä, joka tarkentaa Java-ohjelmointiympäristön tietyn laitteiden joukolle. Konfiguraatio on läheisesti sidoksissa Java virtuaalikoneeseen (JVM), jonka tehtävä pääasiassa on sovelluskoodin suorittaminen. Käytännössä konfiguraatio määrittelee Java-kielen ominaisuudet ja ydinkirjastot. Tyypillisesti konfiguraatio määrittelee

- millaista ja kuinka paljon muistia ohjelmistoympäristölle on varattava
- prosessorityypin ja kellotaajuuden
- näytön
- minkälaiset verkkoyhteydet ovat saatavina.

Konfiguraatio kuvaa vähimmän sovellusalustan kohdelaitteistolle eikä sen ole sallittua määritellä vaihtoehtoisia ominaisuuksia. Laitetoimittajilta vaaditaan määrittelyn täydellistä toteutusta, jotta ohjelmoijat voivat luottaa yhdenmukaiseen ohjelmointiympäristöön. Se mahdollistaa laiteriippumattomien sovellusten toteutuksen. [4; 5.]

Java ME määrittelee kaksi erillistä konfiguraatiota. Connected Limited Device Configuration (CLDC) on kohdennettu kulutuselektoniikan tarpeisiin. Tyypillinen CLDC-sovellusalusta on matkapuhelin tai kämmentietokone, jossa muistia on 512 Kt. CLDC on läheisesti yhteydessä langattomaan Javaan, joka mahdollistaa käyttäjien ostaa ja ladata matkapuhelimeen pieniä sovelluksia - MIDlettejä. Toinen konfiguraatio CDC (Connected Device Configuration) kohdentuu CLDC:n ja työpöytäsovellusten väliselle alueelle. CDC:hen pohjautuvilla laitteilla on muistia tyypillisesti 2 Mt ja monipuolisempi prosessori, jonka ansioista niihin on voitu toteuttaa enemmän Java-ohjelmointikielestä. CDC:tä hyödyntäviä laitteita ovat muun muassa älypuhelimet, web-puhelimet ja kädessä kannettavat tietokoneet. [5.]

Konfiguraatiot koostuvat Java-virtuaalikoneesta sekä kokoelmasta Javan ydinluokkia. Kokoelma Javan ydinluokkia mahdollistaa sovellusohjelmien ohjelmointiympäristön. Prosessori- ja muistirajoituksista johtuen Java ME -virtuaalikoneiden on mahdotona toteuttaa kaikkia Javan ominaisuuksia kuten tavukoodikäskyjä tai koodinsuoritusoptimointeja, jotka normaalisti olisi toteutettu Java SE -ympäristössä. Tavallisesti Java ME -virtuaalikoneet määritellään suppeammaksi osaksi Java-virtuaalikoneen ja Java-kielimääritystä. Java ME -virtuaalikoneen ei ole edellytetty esim. CLDC:ssä tukevan liukulukumuuttujatyyppejä kuten floatia ja doublea.

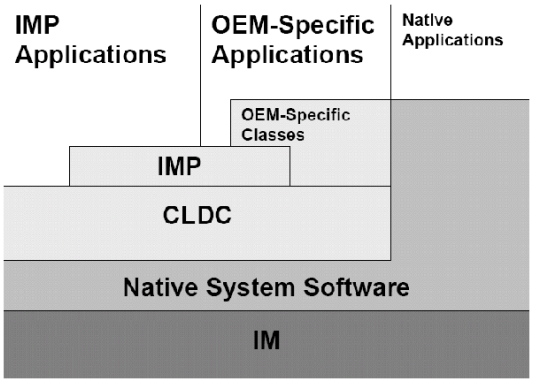
4.2.2 Profiilit

Profiilit täydentävät konfiguraatiota lisäämällä erityisiä ominaisuuksia kirjastoina laitteille tai tietylle markkinasegmentille. Kummallekin Java ME -konfiguraatiosta on yksi tai useampi profiili. Profiilit voivat edelleen laajentaa muita profiileja tuomalla yhä kehittyneempiä ominaisuuksia ohjelmoijan hyödynnettäväksi sovelluksissa. [4; 5.]

Tunnetuin profiili CLDC:lle on Mobile Information Device Profile (MIDP), joka lisää verkkopalvelut, käyttöliittymäkomponentit, ajastimet sekä paikallisen varaston tiedon tallentamiseen. MIDP-profiili on erityisesti suunnattu matkapuhelimiin, joissa on rajoitettu näyttö- ja tallennuskapasiteetti. MIDP on parhaiten tunnettu Java ME:n profiileista, koska se on langattoman Javan (Wireless Java) alusta. [5; 6.]

Information Module Profile (IMP) on tarkoitettu sulautettuihin järjestelmiin, joissa ei ole näyttöä tai mahdollisuutta käyttöliittymän ohjelmoinnille. IMP-profiili pohjautuu MIDP-versioon 1.0. Käytännössä IMP-profiili on osajoukko MIDP-profiilia, josta on karsittu muun muassa käyttöliittymäohjelmoinnin mahdollistava javax.microedition.lcdui pakkaus pois. IMP-profiilin ajoympäristö on monisäikeistystä tukeva ja identtinen ohjelman elikaarelta verrattuna MIDP-profiiliin. IMP-profiili tukee lisäksi tiedon pysyvää tallentamista RMS-rajapinnalla (Record Management System), ajastimia sekä Connector-sovellusohjelmointirajapintaa. Connector-sovellusohjelmointirajapinta mahdollistaa erilaisten yhteyksien luomisen, kuten HTTP- ja pistokeyhteydet (socket). IMP-profiili edellyttää laitteistolta haihtumatonta muistia 128 kilotavua IMP-komponenteille, 8 kilotavua haihtumatonta muistia sovellusten luomalle pysyvälle tiedolle sekä 32 kilotavua haihtuvaa muistia Java-virtuaalikoneen pinolle. [5; 6; 7.]

IMP-NG-profiili (NG - Next Generation) pohjautuu MIDP-versioon 2.0. IMP-NG-profiili julkaistiin marraskuussa 2005 vastauksena kasvavaan tarpeeseen erilaisille sulautetuille verkkoyhteyttä hyödyntäville laitteille. IMP-NG-profiilin lisäominaisuuksia verrattuna IMP-profiiliin ovat tietoturva- ja kommunikointiprotokollalisäykset. IMP-NG-profiiliin on mm. lisätty HTTPS-salattu verkkoprotokolla, sertifikaattivarmenteet, verkkoyhteys hyödyntäen datagrammeja sisältäen salatut pistokkeet (secure socets) sekä automaattinen ohjelmiston etäpäivitystoiminto (over-the-air-provisioning). [7.]



Kuva 6. IMP-profiilin arkkitehtuuri [7]

Kuvassa 6 näkyy IMP-profiilin sekä siihen yleisimmin liitettyjen apukirjastojen sijoittuminen arkkitehtuurikaaviossa. Alimmalla tasolla on laitteisto, jonka päällä sijaitsee käyttöjärjestelmä. Käyttöjärjestelmän päällä on CLDC-konfiguraatio, joka sisältää mm. Java-virtuaalikoneen. IMP-profiili sijoittuu CLDC-konfiguraation päälle yhdessä OEM-spesifisten luokkien kanssa. Koska IMP-profiili on tarkoitettu alustariippumattomaksi, sen osoiteavaruus ei kykene osoittamaan kaikkialle ja siksi tarvitaan OEM-spesifiset luokat toteuttamaan halutut lisäominaisuudet IMP-profiilia hyödyntävien sovellusten tueksi. [7.]

PDA-profiili (PDAP) on samankaltainen MIDP:n kanssa. Se on kohdennettu kämmentietokoneiden tyyppisille laitteille, joissa on isompi näyttö ja enemmän muistia kuin matkapuhelimissa.

Foundation Profile pohjautuu CDC-konfiguraatioon ja sisältää lähes kaikki ydinkirjastot Java 2 -versiosta 1.3. Personal Basis ja Personal Profiles lisäävät Foundation-profiiliin käyttöliittymäkirjastoja. Personal Basis on tarkoitettu yhden aktiivisen ikkunan sovelluksiin ja Personal Profiles tuo tuen monimutkaisten käyttöliittymien rakentamiseen. RMI-profiili lisää Java SE -etäfunktioiden kutsumistuen (Remote Method Invocation) Foundation-profiiliin. Game Profile on myös CDC-konfiguraatioon pohjautuva profiili, joka lisää sovellusalustaan paremman tuen pelien ohjelmoinnille. [5; 6.]

4.2.3 Java ME -arkkitehtuuri

Java ME voidaan esittää arkkitehtuuriltaa kuvan 7 mukaisena kerrosrakenteena. Alimmaisena on käyttöjärjestelmäkerros, joka tarjoaa virtuaalikoneelle toimintaympäristön. Virtuaalikone pohjautuu joko CDC- tai CLDC-konfiguraatioon. Konfiguraatioon voidaan liittää erillinen profiili tukemaan sovellusohjelman vaatimia tarpeita.

Kuva 7. Yleinen Java ME -arkkitehtuuri [4]

Kuva 8. MIDP-profiilin arkkitehtuuri [4]

Kuvassa 8 esitetään MIDP-arkkitehtuurin kerrokset. Alimmalla tasolla tarvitaan käyttöjärjestelmä, jonka päällä virtuaalikonetta ajetaan. KVM (Kilobyte Virtual Machine) -virtuaalikone on Sunin sovittama Java-virtuaalikone (Java Virtual Machine - JVM) pohjautuen CLDC-konfiguraation määrityksiin. KVM ei ole ainoa MIDP-arkkitehtuurin virtuaalikone vaan laitevalmistaja voi liittää oman virtuaalikoneensa tai ostaa sen toiselta yritykseltä. Esimerkiksi IBM:llä on J9 VM -virtuaalikone MIDP-profiilille.

4.3 Java verrattuna perinteisiin sulautettujen järjestelmien toteutuskieliin

Perinteisissä sulautettujen järjestelmien ohjelmointikielissä kuten C:ssä ja C++:ssa sovellusohjelmoijan on tunnettava tarkkaan laitteistoympäristö, johon ohjelmaa kirjoitetaan. Ohjelmissa viitataan suoraan muistiosoitteisiin ja tulee tarkasti tietää, mitä mihinkin muistin alueeseen on laitettu, jotta vältytään tietojen ylikirjoittamiselta. Java-ohjelmoijalla ei ole mahdollisuutta osoittaa muistialueita suoraan ohjelmakoodissaan, vaan siitä huolehtii virtuaalikone. Virtuaalikone helpottaa huomattavasti Java-ohjelmoijan työtä, koska se hallitsee kaikki muistiviittaukset automaattisesti. Javalla ohjelmoitaessa sulautettua järjestelmää ohjelmoiminen on yksinkertaisempaa ja nopeampaa, koska osan ajattelutyöstä voi huoletta jättää virtuaalikoneen hallittavaksi.

Java lähestyy sulautettuja järjestelmiä näkökulmasta, jossa sovellusohjelmoija jo nykyisellä Java-tietämyksellään pystyy kirjoittamaan sovelluksen tuntematta tarkasti laitteistoympäristön kaikkia yksityiskohtia. Jo peruskäsitteet Javasta ymmärtävä pystyy kirjoittamaan ohjelmia sulautettuun Java-ympäristöön. Tämä lisää sovelluskehittämisen nopeutta, koska voidaan suoraan hyödyntää olemassa olevia kirjastoja sekä tietotaitoa ohjelmointikielestä. [6.]

Kovia reaaliaikavaatimuksia tavoiteltaessa joudutaan Javasta toistaiseksi luopumaan sulautettujen järjestelmien ohjelmointikielenä. Koska Javalla toteutettu ohjelmakoodi menee prosessorille suoritukseen virtuaalikoneen kautta, se aiheuttaa viivettä verrattuna Assembler-,

C- tai C++-kääntäjän tuottamaan binäärisen koodin suoritusnopeuteen. Javan hitaudesta johtuen tarvitaan yleensä reaaliaikaisuuden toteuttamiseen rinnakkainen apuprosessori, joka on varsinaisessa yhteydessä luettaviin tai kirjoitettaviin laitteisiin. Java-ohjelma voi näin olla jatkuvassa yhteydessä apuprosessoriin, joka välittää sille viimeisimpiä tietoja.

4.4 UML-mallinnus (Unified Modeling Language)

Ajoneuvotietokoneen FLSender-ohjelmiston suunnittelussa on hyödynnetty vahvasti oliopohjaisuutta. Javan ansiosta sulautetussa ajoneuvotietokoneen ohjelmistossa saadaan oliopohjaiset suunnittelumenetelmät käyttöön. Suunnittelu nojaa UML (Unified Modeling Language) -kieliseen mallinnukseen, joka on alun perin kehitetty järjestelmä- ja ohjelmistokehitystä varten. UML syntyi Object Management Groupin (OMG) vuonna 1997 tekemänä graafisen mallinnuskielen standardina yhdistämällä kolme johtavaa oliomallinnustekniikkaa (OMT, Booch, OOSE). Uusin UML-standardi on 2.0 vuodelta 2004. UML-kieli sisältää 13 erilaista kaaviota, joista kuudella kuvataan ohjelmiston rakennetta, kolmella käyttäytymistä ja neljällä vuorovaikutusta. UML-kaaviot havainnollistavat ohjelmistoa erilaisilla abstraktiotasoilla. Niiden avulla päästään parempaan ja laadukkaampaan suunnittelutulokseen. Tässä työssä hyödynnetään viittä erilaista UML-kaaviota: käyttötapaus-, luokka-, pakkaus-, sijoittelu- ja sekvenssikaaviota. [15.]

Käyttötapauskaavio (use case diagram) on yksi UML-mallinnuskielen kaavioista, joka kuvaa ohjelmiston käyttäytymistä. Korkean tason käyttötapauskaaviolla hahmotetaan ohjelmiston tärkeimmät toiminnot ja sen pohjalta ohjelmoija voi lähteä abstrahoimaan oleellisia olioita ja niiden suhteita toisiinsa. Käyttötapauskaaviolla on myös muita merkityksiä. Käyttötapausten ja käyttäjien määrä sekä niiden keskinäinen suhde toisiinsa kuvaa järjestelmän kompleksisuutta. Mitä enemmän on käyttötapauksia ja käyttäjiä, sitä enemmän aikaa täytyy varata järjestelmän suunnitteluun. Insinöörityössä on laadittu FLSender- ja FLReceiver-sovellusten käyttötapauskaaviot. FLSender-sovelluksen käyttötapauskaavio on kuvassa 19 (s. 32) ja FLReceiver-sovelluksen käyttötapaukset ovat esitetty kuvassa 34 (s. 47).

Luokkakaavio (class diagram) on ohjelmiston rakennekaavio, joka kuvaa kuinka luokat ovat yhteydessä toisiinsa ja mitä muuttujia ja funktioita luokat sisältävät. Luokat ovat olio-ohjelmoinnin elementtejä, joista koko ohjelmisto rakentuu. Luokkakaavion tekeminen on yksi oliosuunnittelun tärkeimmistä vaiheista. Luokkakaavio vaikuttaa suoraviivaisesti ohjelmoijan kirjoittamaan koodiin. Uusimmat UML-ohjelmistot osaavat tuottaa jopa valmista

ohjelmakoodia laaditun luokkakaavion pohjalta. FLSender-sovelluksen luokkakaavio on laitettu liitteeksi 1 ja FLReceiver-sovelluksen luokkakaavio liitteeksi 3.

Pakkauskaavio (package diagram) on järjestelmän rakennekaavio, joka kuvaa kuinka ohjelmisto on pilkottu loogisiksi ryhmiksi/osiksi. Kaavion sisältämät pakkaukset ovat UML-rakenteita, joilla kuvataan erilaisten yhteen kuuluvien elementtien ryhmittymiä. Pakkauskaavio esittää myös ohjelmiston pakkausten väliset riippuvuudet. Kaavion sisältämät pakkaukset on usein esitetty tiedostokansioina. Pakkauksia voi käyttää missä tahansa UML-kaaviossa. Yleisimmin pakkauksia käytetään luokkien tai käyttötapausten kokoamiseen ryhmiksi. Pakkausten tarkoitus on koota isoja kokonaisuuksia yhteen, jolloin kaavioista tulee yksinkertaisempia ja helmpompia ymmärtää. FLSender-sovelluksen pakkauskaavio on esitetty kuvassa 20 (s. 33).

Sijoittelukaavio (deployment diagram) on korkean tason rakennekaavio, joka kuvaa järjestelmän teknistä arkkitehtuuria esittämällä toteutuksessa käytetyn laitteiston (hardware), komponenttien sijoittelun laitteistolle sekä väliohjelmiston (middleware), jolla erilaisissa laitteistoissa olevat komponentit liittyvät toisiinsa. Sijoittelukaavio ilmaisee myös toisiinsa liittyvien komponenttien väliset suhteet. Sulautetuissa järjestelmissä sijoittelukaaviota käytetään esittämään järjestelmän arkkitehtuuria sekä kuinka laitteisto ja ohjelmistokomponentit toimivat yhdessä. Kuvan 17 (s. 26) sijoittelukaavio kuvaa asiakassovelluksen (FLSender) sijoittumisen Aplicom A1 Flex -ajoneuvotietokoneen laitteistossa sekä tämän kommunikoinnin ja suhteet muihin kompomentteihin.

Sekvenssikaavio (sequence diagram) kuvaa olioiden vuorovaikutusta ajan kuluessa. Se kuvaa vierekkäisinä pystysuorina viivoina samaan aikaan elävien olioiden elämänviivat ja vaakasuuntaisina nuolina niiden välillä kulkevat viestit tapahtumajärjestyksessä. Aika kulkee kaaviossa ylhäältä alaspäin. Sekvenssikaavion avulla voidaan helposti havainnollistaa ohjelmiston jokin toiminnallisuus. FLSender-ohjelmiston osista on laadittu useita sekvenssikaavioita. Niitä ovat esimerkiksi tapahtumienkäsittelyn (kuva 25, s. 39), digitaalisen auran tapahtuman (kuva 26, s. 40), tietojen tallentamisen (kuva 28, s. 42) ja tietojen lähettämisen palvelimelle (kuva 30, s. 43) havainnollistamat kaaviot.

4.5 Suunnittelumallien hyödyntäminen sulautetussa ohjelmistossa

Suunnittelumallit ovat kuvauksia keskenään vuorovaikutuksessa olevista olioista ja luokista, jotka on muotoiltu ratkaisemaan tietyssä yhteydessä esiintyvä yleinen suunnitteluongelma. Java ME tarjoaa täysin oliopohjaisena ohjelmointikielenä kehittäjille mahdollisuuden soveltaa yleisiä olio-pohjaisia suunnittelumalleja ohjelmistojen toteutuksessa. Suunnittelumalleja käyttämällä voidaan uudelleenkäyttää hyväksi havaittuja ratkaisuja ja arkkitehtuureja. Suunnittelumallit auttavat valitsemaan vaihtoehtoja, jotka tekevät ohjelmistosta uudelleenkäytettävän ja välttämään ratkaisuja, jotka vähentävät uudelleenkäytettävyyttä. Suunnittelumallit parantavat myös olemassa olevien järjestelmien dokumentaatiota ja ylläpidettävyyttä tarjoamalla eksplisiittisen määrityksen luokkien ja olioiden suhteista ja niiden käyttötarkoituksesta. Suunnittelumalli nimeää, abstrahoi ja määrittää yleisen suunnittelurakenteen avainkohdat, jotka hyödyttävät uudelleenkäytettävän oliopohjaisen suunnitteluratkaisun luomista. [8.]

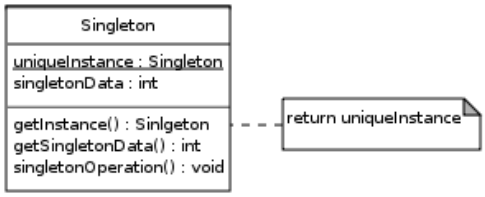
Nykyisten oliosuunnittelumallien isänä voidaan pitää Erich Gammaa, joka esitteli ne väitöskirjassaan vuonna 1991. Myöhemmin Gamma kokosi nk. Gang of Four -ryhmän kanssa teoksen Design Patterns – Elements of Reusable Object-Oriented Software. Se on edelleen tunnetuin suunnittelumalleja käsittelevä teos ja sisältää 23 erilaista suunnittelumallia.

Oliopohjaista suunnittelua pidetään yleisesti ottaen haastavana ja vielä haastavampaa on suunnitella uudelleenkäytettävä oliopohjainen ohjelmisto. On löydettävä sovellusalueen keskeiset oliot, muodostettava niistä luokat sopivalla abstaktiotasolla, määriteltävä luokkien väliset suhteet sekä luokkien rajapinnat ja periytymishierarkiat. Suunnittelun tuloksen on oltava kohdealuespesifinen ja riittävän yleinen, jotta myöhemmät muutostarpeet voidaan joustavasti toteuttaa. Muunneltavan ja uudelleenkäytettävän ratkaisun löytäminen ensi yrittämällä on vaikeaa, ellei jopa mahdotonta. Suunnittelun kuluessa ratkaisua yleensä muutetaan useaan kertaan. Monista oliopohjaisista järjestelmistä löytyy samankaltaisia luokkarakenteita ja olioiden välisiä vuorovaikutustapoja. Hyväksi havaittuja ratkaisumalleja sovelletaan tiettyihin suunnitteluongelmiin ja niiden avulla suunnitteluratkaisuista tulee joustavampia, taidokkaampia ja viime kädessä myös uudelleenkäytettäviä. Ratkaisumallien avulla suunnittelijat pystyvät uudelleenkäyttämään hyväksi havaittuja ratkaisuja ja perustamaan uudet suunnitteluratkaisut aikaisemmalle kokemukselle. Suunnittelija, joka tuntee mallit, pystyy suoraan soveltamaan niitä omassa työssään, eikä hänen tarvitse keksiä samoja ratkaisuja useaan kertaan. [8.]

Perinteisissä proseduraalisissa sulautettujen järjestelmien toteutuskielissä kuten C:ssä ei ole olionäkökulmaa, vaan ohjelmakoodi etenee enemmänkin rivi riviltä alaspäin. Oliopohjaisena ohjelmointikielenä Java ME tuo uudenlaisen näkökulman sulautetun ohjelmiston suunnittelulle. Java-ohjelmistoarkkitehti pääsee suoraan hyödyntämään tuntemiaan oliopohjaisia suunnittelumalleja sulautetussa ympäristössä. Insinöörityön toteutusvaiheessa havaittiin suunnittelumallien sovellettavuus FLSender- ja FLReports-komponenttien toteuttamisessa. Neljää erilaista suunnittelumallia on hyödynnetty kokonaan tai osittain ohjelmiston osien toteutuksissa. Seuraavissa luvuissa esitellään ohjelmiston eri osissa käytetyt suunnittelumallit ja kerrotaan millaiseen tilanteeseen malli tuo apua.

4.5.1 Ainokainen (Singleton)

Joskus on tarve luoda luokasta vain yksi ilmentymä. Järjestelmässä voi olla esimerkiksi useita kirjoittimia, mutta kaikki kirjoittimet jakavat saman tulostusjonon. Muita esimerkkejä ovat tiedostojärjestelmä ja ikkunoinnin hallintajärjestelmä. Olioiden luontimallina Ainokainen varmistaa, että luokasta luodaan vain yksi ilmentymä. Se tarjoaa globaalin tavan päästä käsiksi ilmentymään sekä estää useiden ilmentymien luonnin. [8.]

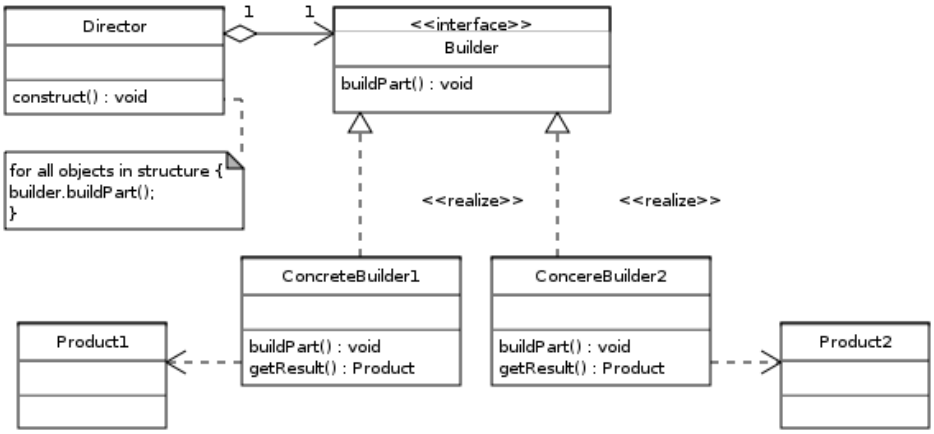


Kuva 9. Luokkakaavio Singleton-suunnittelumallista

Kuvan 9 luokka toteuttaa ainokainen suunnittelumallin. Siinä luokan staattinen jäsenmuuttuja uniqueIntsnace pitää sisällään viitettä, jota pyytäjille jaetaan. Ainoa globaali tapa hakea luokasta ilmentymä on kutsua staattista metodia getInstance. Kutsumalla metodia ensimmäisen kerran luodaan luokasta ilmentymä ja seuraavilla kerroilla getInstance palauttaa jo luodun olion viitteen. Luokan rakentaja (constructor) tulee asettaa private-määreen alle, jolla estetään monien ilmentymien luonti. Singleton suunnittelumallia tulee käyttää, kun luokalla täytyy olla täsmälleen yksi ilmentymä ja sen täytyy olla kaikkialla sovelluksessa saatavilla.

4.5.2 Rakentaja (Builder)

Rakentaja-malli on olioden luontimalli, joka erottaa toisistaan monimutkaisen olion rakentamisessa käytetyn prosessin ja olion esitysmuodon. Samalla rakentamisprosessilla voidaan tuottaa erilaisia esitysmuotoja. Rakentaja-malli kuvaa ratkaisun periaatteen. Malli sopii hyvin tilanteeseen, jossa algoritmi halutaan pitää erillään monimutkaisesta osista koostuvasta olion luonnista: osista, joista olio koostuu, sekä tavasta, jolla osat yhdistetään. Rakentaja-mallia tulisi myös käyttää tilanteessa, jossa rakentamisprosessin on pystyttävä tuottamaan esitysmuodoltaan erilaisia olioita. [8.]

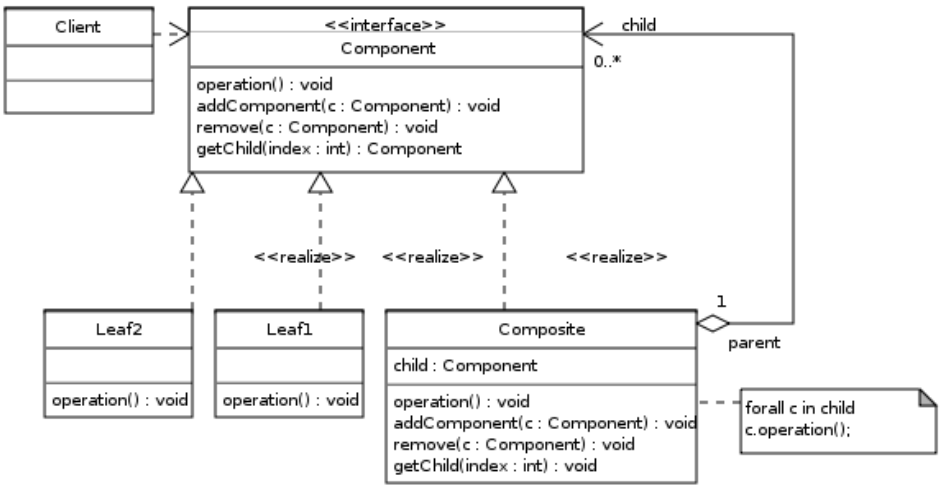


Kuva 10. Luokkakaavio Builder-suunnittelumallista

Kuvassa 10 luokka Builder määrittelee abstraktin rajapinnan, jolla luodaan tuotteiden eli Product-olioiden osat. Kongreettiset rakentajaluokat (ConcreteBuilder1 ja ConcreteBuilder2) toteuttavat Builder-rajapinnan ja käyttävät sitä tuotteen rakentamiseen osista kokoamalla. Kongreettiset rakentajat pitävät kirjaa tuotteen rakenteen edistymisestä rakentamisen kuluessa. Ne myös toteuttavat tuotteen hakuoperaatiot getResult-metodissa. Director on apuluokka, joka kokoaa olion käyttäen Builder-rajapintaa. Luokka Product kuvaa rakentamisen kohteena olevan tuotteen. ConcreteBuilder rakentaa tuotteen sisäisen esitysmuodon ja määrittelee prosessin, jonka mukaisesti kokoaminen tehdään. Product-luokka sisältää myös luokat, jotka määrittelevät rakennettavan olion osat ja operaatiot, joilla osista kootaan lopullinen tulos. [8.]

4.5.3 Rekursiokooste (Composite)

Grafiikkasovelluksissa esimerkiksi graafisissa editoreissa ja kaaviojärjestelmissä käyttäjä voi rakentaa monimutkaisia kaavioita yksinkertaisista osista. Käyttäjä kokoaa osat suuremmaksi kokonaisuudeksi, jotka edelleen yhdistetään suuremmiksi kokonaisuuksiksi. Suoraviivaisessa toteutuksessa määriteltäisiin graafisille primitiiveille omat luokat ja niiden lisäksi säiliöluokat, jotka sisältävät primitiivejä. Ratkaisu on kuitenkin ongelmallinen, sillä primitiiviluokkia ja säiliöluokkia pitäisi käsitellä koodissa eri tavalla, vaikka käyttäjä käsittelee molempia useimmiten samalla tavalla. Sovelluksesta tulee monimutkaisempi, jos oliot on pakko erotella toisistaan. Rekursiokooste-malli neuvoo, miten rekursiivista koostetta käytetään siten, että sovelluksessa ei tarvitse tehdä tätä erottelua. [8.]

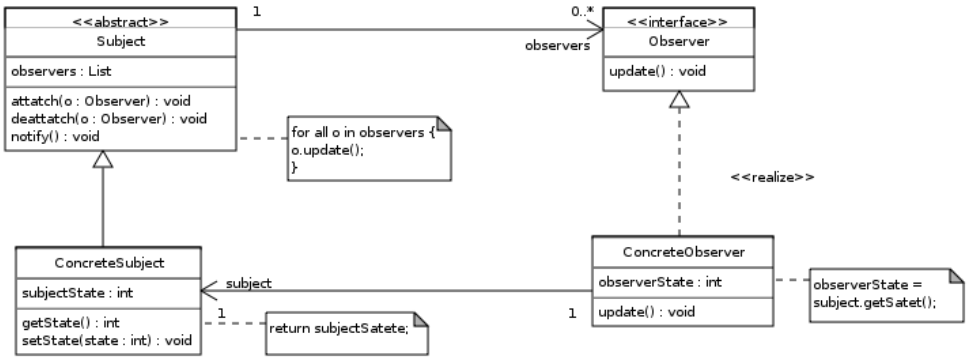


Kuva 11. Luokkakaavio Composite-suunnittelumallista

Rekursiokoostemalli on olioiden rakennemalli, joka esittää oliot rekursiivisesti koostettuna puurakenteena. Yksittäisiä oliota ja oliokoosteita voidaan käsitellä samalla tavalla. Kuvassa 11 on määritelty rajapinta Component, jonka toteuttavat niin lehdet kuin koosteoliokin. Component-rajapinta määrittelee hierarkisen rakenteen kaikkien solmujen yhteiseksi rajapinnaksi, ja se voi antaa yhteisille operaatioille oletustoteutuksen. Component-rajapinta määrittelee myös operaatiot, joilla päästään käsiksi lapsisolmuihin. Composite-luokka kuvaa koosteolion, joka ylläpitää lapsia. Kuvassa 11 on Component-rajapinnan tyyppinen taulukko child, johon lehdet varastoidaan. Lehdet kuvaavat koosterakenteen primitiiviolioita ja kunkin lehden oma konkreettinen toteutus operation-metodista mahdollistaa lehtien toimivan eri tavoin, mutta kaikkia lehtiä käsitellään silti samalla tavalla. Asiakas käsittelee koosterakenteen olioita vain Component-rajapinnan kautta. Koosteoliolle tulleet metodikutsut delegoidaan kaikille child-taulukossa oleville lehdille. [8.]

4.5.4 *Tarkkailija (Observer)*

Jos järjestelmä on jaettu joukoksi yhteistyössä toimivia luokkia, on huolehdittava myös siitä, että olioiden tilat pysyvät keskenään yhtenäisinä. Luokkien yhtenäisyyttä ei haluta varmistaa sitomalla niitä kiinteästi toisiinsa, koska tällöin luokkien uudelleenkäytettävyys huononisi. Tarkkailija-suunnittelumalli määrittelee olioiden välille yksi moneen -riippuvuuden siten, että kun yhden olion tila muuttuu, siitä riippuvat oliot saavat ilmoituksen ja päivittyvät automaattisesti. [8.]



Kuva 12. Luokkakaavio Observer-suunnittelumallista

Kuvan 12 tarkkailijamalli kuvaa oliojoukon käyttäytymistä. Mallin tärkeimmät oliot ovat subjekti (Subject) ja tarkkailija (Observer). Subjektiin voi liittyä mielivaltaisen monta tarkkailijaa. Kaikille tarkkailijoille ilmoitetaan, kun subjektin tilassa tapahtuu muutos. Ilmoituksen seurauksena kukin tarkkailija kysyy subjektilta sen tilatiedot päivittääkseen oman tilansa vastaavasti. Luokassa Subject määritellään rajapinta, jolla kiinnitetään ja irrotetaan tarkkailijoita. Luokassa Observer määritellään rajapinta olioille, joiden halutaan saada ilmoitus subjektin tilassa tapahtuneista muutoksista. ConcreteSubjet tallentaa tilatiedot, jotka kiinnostavat ConcreteObserver-olioita sekä lähettää tarkkailijoilleen tiedon tilassa tapahtuneista muutoksista. ConcreteObserver pitää viitettä ConcreteSubject olioon ja tallentaa tilatiedot, joiden on pysyttävä synkronissa subjektin kanssa. ConcreteObserver toteuttaa Observer-rajapinnan ja käyttää sitä tilatietojensa päivittämisessä. [8.]

5 TEKNINEN TOTEUTUS

5.1 Laitteisto - A1 Flex -ajoneuvotietokone

Raskaan kaluston ajoneuvossa on kuvassa 13 oleva tiedonkeruulaite. A1 Flex on Java-ohjelmoitava tietokone, joka sisältää monipuoliset I/O-liitynnät erilaisten sovellusten toteuttamiseksi. Tyypillisimpiä sovelluksia ovat kuljettajan tilan ja työaikojen seuraaminen reaaliaikaisesti. Myös kuljettajan käyttäytymisen ja ajoneuvosta taloudellisesti merkittävien tietojen tallentaminen on toteutettu tyypillisimmissä sovelluksissa. [9.]

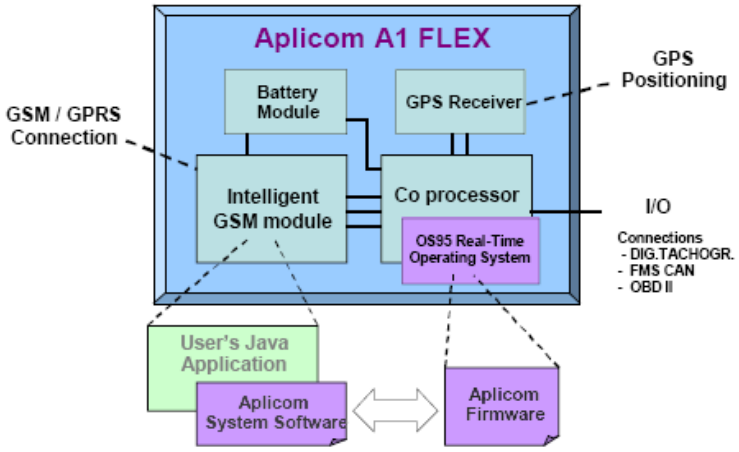
Kuva 13. Ajoneuvossa oleva tietokone A1 Flex

A1 Flexissä ei ole näyttöä tai painikkeita. Laite sisältää kuvassa 13 olevan violettisen suojan takana kuitenkin 4 LEDiä (Light-Emitting Diode), joista 3 on vapaasti ohjelmoitavia. LEDeillä voidaan ilmaista esimerkiksi erilaisia ohjelman tiloja. Kukin LEDeistä voi palaa vihreänä, punaisena tai yhtä aikaa vihreänä ja punaisena. Kaikki I/O-liitynnät on sijoitettu laitteen takaosaan.

5.1.1 A1 Flexin moduulit

Kuva 14 esittää A1 Flexin koostuvan neljästä erillistä moduulista, jotka ovat älykäs GSM-moduuli, reaaliaikainen apuprosessori, GPS-paikkatiedon vastaanotin sekä akkumoduuli. Laitteen Siemens TC65 GSM -moduuli mahdollistaa vapaasti ohjelmoitavat Java imlet -sovellukset Java ME -alustan IMP-NG-profiilin kirjastoin. Siemens TC65 GSM -moduuli tarjoaa lisäksi GSM- ja GPRS-yhteydet.

ARM7-apuprosessori käsittelee reaaliaikaprosessointiin liittyvät tehtävät. Apuprosessorilla on oma reaaliaikainen käyttöjärjestelmä, joka hallitsee mm. I/O-väyliä. GPS-tietojen vastaanotin on 16-kanavainen moduuli, joka on liitetty apuprosessoriin. Apuprosessori hallitsee GPS-moduulia ja päivittää esimerkiksi tarkan kellonajan GPS-moduulilta reaaliaikakelloonsa. GPS-moduuli tukee lisäksi kahta etäisyysmittaria. Akkumoduuli on liitettu GSM- ja apuprosessori-moduuleihin. Akkumoduuli tarjoaa virtaa järjestelmälle silloin, kun päävirtasyöttö katkeaa. [9.]

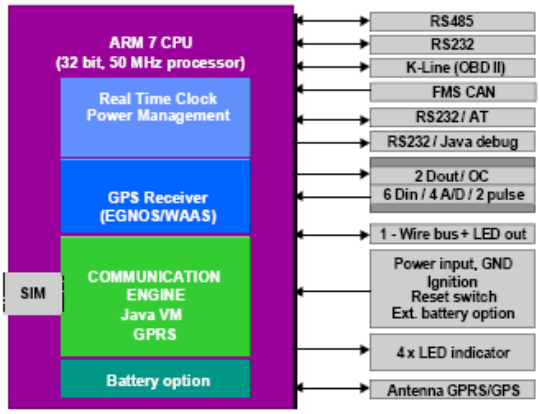


Kuva 14. A1 Flex koostuu älykkäästä GSM-, akku-, GPS- ja apuprosessorimoduulista [9]

Kuva 14 havainnollistaa myös erilaisten moduulien väilisiä yhteyksiä. Siinä Aplicom on liittänyt Siemensin GSM-moduulin apuprosessoriin. GSM-moduulissa suoritetaan FLSender Java imlet -sovellusta (User's Java Application). FLSender kommunikoi apuprosessorin kanssa Aplicomin GSM-moduuliin kehittämän SYSW-järjestelmäkirjaston (Aplicom System Software) avulla. Apuprosessorilla on reaaliaikainen käyttöjärjestelmä (OS95 Real-Time Operating System), jonka Aplicom on sovittanut GSM-moduulin järjestelmäkirjastoon omalla laiteohjelmalla (Aplicom Firmware).

5.1.2 A1 Flexin I/O-liitynnät

A1 Flex sisältää monipuoliset I/O-liitynnät erilaisten laitteiden ja väylien liittämiseen laitteeseen. Kuvassa 15 näkyy apuprosessorin kommunikointi varsinaisten laite-I/O väylien kanssa. Apuprosessori välittää laitteille lähetettävät tai laitteilta luettavat tiedot Siemensin GSM-moduulin Java-ohjelmalle. Apuprosessorin reaaliaikainen käyttöjärjestelmä mahdollistaa tietojen jatkuvan lukemisen ilman liian suuria viiveitä.



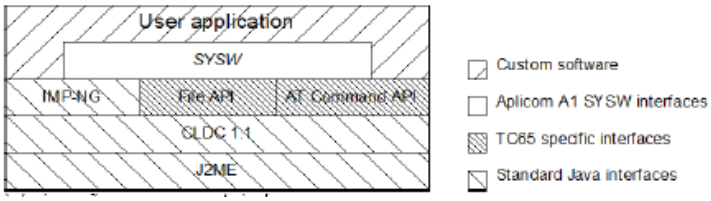
Kuva 15. A1 Flexin I/O-liitynnät [9]

A1 Flex on mahdollista kytkeä moniin väyliin ja laitteisiin. RS485- ja RS232-väylät ovat vapaasti ohjelmoitavia väyliä, joihin voidaan kirjoittaa ja lukea. Kierretyllä parikaapelilla toteutettua RS485-väylää käytetään tyypillisesti teollisuudessa liittämään yhteen väylään useita laitteita. RS485-väylään liitetyt laitteet voivat sijaita pitkänkin etäisyyden päässä. Suurin etäisyys on 1200 metriä. RS232-väylä on kahden tietokonelaitteen väliseen tietoliikenteeseen tarkoitettu tietoliikenneportti, jossa tieto siirtyy yksi bitti kerrallaan "peräkkäin" sarjamuotoisena. [10; 11.]

K-Line-väylä eli OBD II on tarkoitettu erilaisen diagnostiikkatiedon lukemiseen ajoneuvosta. FMS-CAN-väylä on sähköinen väylä, jota käytetään ajoneuvoissa, koneissa ja teollisuuslaitteissa. Väylältä saadaan esimerkiksi raskaan kaluston ajoneuvon moottorin tietoja. A1 Flexissä on 6 digitaaliseen sisääntuloon tarkoitettua kanavaa. Niistä saadaan yksinkertainen binäärinen tieto signaalin tilasta. Laite tukee myös signaalin leveyden mittaamista.

5.1.3 A1 Flexin ohjelmointirajapinnat

Kuva 16 havainnollistaa A1 Flexin sisältämiä Java-kirjastoja. Standardeja Java-kirjastoja ovat J2ME, CLDC1.1 ja IMP-NG. IMP-NG (Information Module Profile - Next Generation) on osajoukko MIDPtä (Mobile Information Device Profile). IMP-NG:stä on jätetty pois käyttöliittymäohjelmoinnin mahdollistava pakkaus, jota ei aina tarvita sulautetuissa järjestelmissä. Java-sovelluksia, jotka pohjautuvat IMP-profiiliin kutsutaan imleteiksi vaikka käytännössä niitä voitaisiin hyvin kutsua midleteiksi, koska IMP-profiili on osajoukko MIDPtä. [12.]



Kuva 16. A1 Flexin Java-ohjelmointirajapinnat [12]

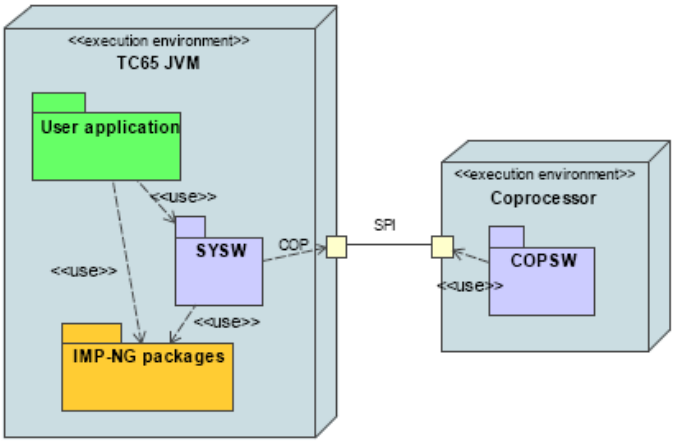
A1 Flex sisältää standardien Java-kirjastojen lisäksi kahden laitetoimittajan omia kirjastoja. Siemens on toteuttanut tiedostojen käsittelyyn kirjaston (File API) ja AT-modeemikomentoihin (AT Command API) tarvittavan kirjaston. Aplicom on lisännyt oman SYSW-järjestelmäkirjaston (System Software), jolla liitytään apuprosessoriin. [12.]

Siemensin tiedostojen käsittelyn mahdollistava kirjasto (File API) tarjoaa helpon tavan kirjoittaa laitteen FLASH-tiedostojärjestelmään. GSM-moduulissa sovelluksen käyttöön varattua FLASH-muistia on noin 1,7 Mt. FLASH-tiedostojärjestelmä rajoittaa tiedostoviittauksen pituudeksi 124 merkkiä ja suositeltavien tiedostojen ja alihakemistojen määrä hakemistoa kohden on 200 kappaletta. Toinen Siemensin tarjoama kirjasto on AT-modeemikomentojen toiminnallisuuden toteuttava AT Command API. Sillä käyttäjä voi lähettää GSM/GPRS-modeemille AT-komentoja. AT-komennot ovat suora käyttöliittymä modeemiin. Niillä voidaan muodostaa yhteys, katkaista yhteys ja konfiguroida monia modeemin asetuksia. [13; 14.]

Aplicomin SYSW-järjestelmäkirjasto mahdollistaa käyttäjäsovelluksen kommunikoinnin apuprosessorin kanssa, hallitsee konfiguraatioita, apuprosessorin sarjaportin kommunikoinnin, paikkatiedon lukemisen ja kaiken käyttäjäsovelluksen I/O:n lukemiseen ja kirjoittamiseen tarvittavat toiminnot. Se myös hallitsee kummankin GSM-moduulin ja apuprosessorin virranhallinnan. Aplicomin järjestelmäkirjasto sisältää lisäksi hyödyllisiä palveluita, joita ei normaalisti ole käytössä Java ME -ympäristössä. Tällainen on esimerkiksi Properties-luokka, joka tunnetaan Java SE:ssä Property-luokkana. [12.]

5.1.4 Java imlet -asiakassovelluksen kommunikointi apuprosessorin kanssa

Kuvassa 17 on UML-sijoittelukaavio, joka havainnollistaa asiakassovelluksen (user application) kommunikointia apuprosessorin kanssa. Siemensin TC65 GSM -moduulin Java-virtuaalikoneessa ajettava imlet-asiakassovellus käyttää Aplicomin SYSW-järjestelmäkirjastoa liittyäkseen sarjaväylän kautta apuprosessorin sulautettuun COPSW-sovellukseen (COprocessor SW). Sulautettu COPSW-sovellus kontrolloi apuprosessoria ja ottaa vastaan asiakassovelluksen lähettämät pyynnöt, käsittelee ne ja paulauttaa asiakassovellukselle tietoja. IMP-NG-pakkaukset ovat standardeja imlet-kirjastoja, joita asiakassovellus käyttää.

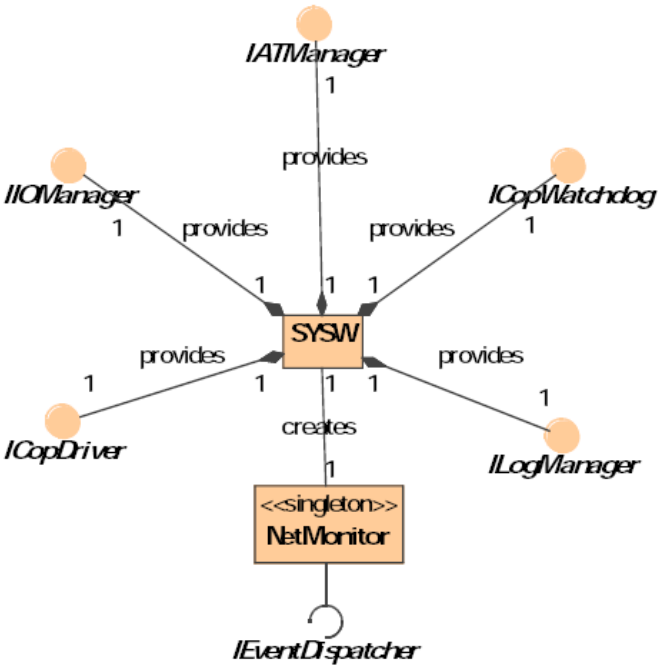


Kuva 17. UML-sijoittelukaavio asiakassovelluksen kommunikoinnista apuprosessorin kanssa [12]

SYSW-järjestelmäkirjasto on liitetty nopean synkronisen sarjaväylän (SPI) kautta apuprosessorin COPSW-sovellukseen. Asiakassovelluksen käyttämä SYSW-kirjasto ja COPSW-sulautettu sovellus toteuttavat COP-kommunikointiprotokollan (COprocessor Protocol), joka mahdollistaa apuprosessorin tilan lukemisen ja kaikkien apuprosessoriin liitettyjen I/O-laitteiden käsittelyn. Asiakassovelluksessa kaikki sarjaväylään kohdistuvat pyynnöt ovat läpinäkyviä eli niitä Java-ohjelmoija ei näe millään tavalla. SYSW-kirjasto sisältää monenlaisia luokkia, joilla konfiguroidaan, luetaan ja kirjoitetaan apuprosessoriin liitettyihin I/O-laitteisiin. [12.]

5.1.5 *Asiakassovelluksen käyttämä SYSW-järjestelmäkirjasto*

Kuva 18 esittää luokkakaaviota SYSW-järjestelmäkirjastosta. Luokka Sysw on kirjaston keskeisin pääluokka, joka on toteutettu ainokainen (singleton) suunnittelumallilla. Näin mahdollistetaan yhden viitteen jakaminen helposti kaikkiin asiakassovelluksessa käytettyihin luokkiin. Alustamisen jälkeen Sysw-luokka tarjoaa useita hyödyllisiä palveluita asiakassovelluksen käyttöön. Palvelujen avulla asiakassovellus esimerkiksi alustaa erilaisia apuprosessoriin liitettyjä laitteita (GPS-vastaanotin, CAN-väylä, jne.) ja kommunikoi apuprosessorin kanssa hakeakseen laitteilta jatkuvasti tietoja. Palvelut on kuvattu kuvassa 18 rajapintoina, joista Sysw-luokka koostuu. Sysw-luokan käyttäminen ei ole pakollista, mutta kuitenkin suositeltavaa, koska se alustaa joitakin apuprosessorin ja TC65 GSM -moduulin asetuksia. [12.]



Kuva 18. SYSW-kirjaston luokkakaavio [12]

Koodilistauksessa 1 rivillä 2 asiakassovellus pyytää järjestelmäkirjaston Sysw-luokalta viitettä itselleen. Viitteen saamisen jälkeen rivillä 5 asiakassovellus lähettää järjestelmäkirjastolle alustuspyynnön. Rivillä 8 asiakassovellus pyytää ICopDriver-luokan ilmentymää itselleen järjestelmäkirjaston pääluokan (Sysw) avulla. ICopDriver-oliolla asiakassovellus voi lähettää monenlaisia käskyjä apuprosessorille. Rivillä 11 asiakassovellus pyytää jälleen järjestelmäkirjaston pääluokan (Sysw) avulla lokituspalvelua, jota käytetään esimerkiksi ohjelmiston kehitysvaiheessa debug-tarkoituksiin. Riveillä 14 ja 15 asiakassovelluksessa luodaan komento-olio (CopPosStatusRead) GPS-tietojen luentaa varten apuprosessorilta. Olion konstruktori parametroidaan hakemaan viimeisin paikkatieto. Rivillä 16 lähetetään luotu komento-olio ICopDriver-tyyppisellä oliolla apuprosessorille. Komento-oliolta pyydetään rivillä 17 IPosition-tyyppinen olio pos, joka sisältää GPS-paikkatiedot. Riveillä 18 ja 19 pyydetään pos-oliolta leveysasteet ja pituusasteet double-tyyppisiin muuttujiin.

Koodilistaus 1. SYSW-järjestelmäkirjaston palvelujen käyttäminen asiakassovelluksessa

```
1  // Get instance of System Software (SYSW) singleton
2  Sysw m_sysw = Sysw.getInstance();
3
4  // Initialize SYSW service, and create watchdog
5  m_sysw.initialize(true);
6
7  // Get CopDriver
8  ICopDriver m_copDriver = m_sysw.getCopDriver();
9
10 // Get logger object (singleton)
11 Log m_log = m_sysw.getLogManager().getLogger();
12
13 // Read current position
14 CopPosStatusRead cmd = new
15           CopPosStatusRead(CopPosStatusRead.CMD_CURRENT_POS);
16 m_copDriver.sendCommand(cmd);
17 IPosition pos = cmd.getPosition();
18 double latitude = pos.getLatitude();
19 double longitude = pos.getLongitude();
```

SYSW-järjestelmäkirjasto toteuttaa tapahtumien ilmoittamisen perustuen vahvasti tarkkailija-suunnittelumalliin (observer design pattern). Järjestelmäkirjasto valvoo laitteiden tilaa ja reagoi apuprosessorin tapahtumiin. Sovellusohjelmoijan tehtäväksi jää ohjelmoida konkreettiset tapahtumankuuntelijaluokat, jotka rekisteröidään järjestelmäkirjastolle. Järjestelmäkirjasto jakelee apuprosessorilta saapuneen tapahtuman siihen liitetylle tapahtumankuuntelijalle. Tärkeimmät järjestelmässä generoituvien tapahtumien lähteet ovat kuvan 18 rajapintoja ICopDriver ja IIOManager toteuttavat järjestelmäkirjaston konkreettiset luokat sekä luokka NetMonitor. [12.]

Koodilistaus 2. Asiakassovellukseen ohjelmoidun tapahtumankuuntelijan rekisteröinti SYSW-kirjastolle

```
1 // Create an event listener and register it
2 GeneralEventListener m_listener = new GeneralEventListener(m_timer, this);
3 m_copDriver.addEventListener("exampleApp", m_listener);
```

Koodilistauksessa 2 asiakassovelluksen rivillä 2 luodaan ilmentymä itse ohjelmoidusta tapahtumankuuntelijaluokasta GeneralEventListener. Rivillä kolme rekisteröidään luotu tapahtumienkuuntelijaolio järjestelmäkirjastolle ICopDriver-tyyppisen olion addEventListener-metodilla. Tämän jälkeen järjestelmäkirjasto ilmoittaa itsenäisesti rekisteröidylle oliolle tapahtumista.

Suurimman osan tapahtumista tuottaa SYSW-järjestelmäkirjaston rajapinnan ICopDriver toteuttavat konkreettiset luokat. Ne synnyttävät esimerkiksi uuden tapahtuman, kun laitteen tila vaihtuu. ICopDriver-rajapintaa toteuttavat konkreettiset luokat vastaavat esimerkiksi GPS-paikkatiedon validiteetin-, sisä- ja ulostulojen tilojen-, sarjaportissa valmiina olevan tiedon-, virran tilan vaihtumisen- ja FMS-CAN-tapahtumien synnyttämisestä. Rajapinta ICopDriver määrittelee myös asiakassovelluksen kommunikoinnin apuprosessorin kanssa. Se tarjoaa asiakassovellukselle mahdollisuuden lähettää komentoja ja vastaanottaa apuprosessorin generoimia tapahtumia, jotka se jakelee tapahtumiin liitetyille kuuntelijoille. [12.]

Rajapinta IIOManager määrittelee metodit TC65 GSM -moduulin sisääntulojen tarkkailuun, ja sitä toteuttavat luokat generoivat tapahtuman sisääntulojen vaihtumisesta. SYSW-järjestelmäkirjasto käyttää sisäisesti näitä tapahtumia, eivätkä ne yleensä ole asiakassovelluksen ohjelmoijalle kiinnostavia. [12.]

NetMonitor-luokka on ainokainen (singleton) suunnittelumallilla toteutettu verkkoyhteyden valvontaan tarkoitettu luokka. Sen avulla on mahdollista seurata GSM/GPRS-verkon tilaa. NetMonitor synnyttää tapahtumia verkon tilan vaihtumisesta ja jakelee ne tapahtumiin liitetyille tapahtumankuuntelijoille. [12.]

SYSW-järjestelmäkirjasto toteuttaa ja käyttää pitkälle kehitettyä lokituspalvelua. Lokituspalvelu on kontrolloitu ILogManager-rajapinnan kautta, joka on myös asiakassovelluksen käytettävissä. Oletusarvoisesti lokitustiedot ovat lähetetty A1 Flex -laitteen debug-porttiin, jota voidaan seurata PC:n terminaaliohjelmalla. Vaihtoehtoisesti lokitustiedot on mahdollista tallentaa A1 Flex -laitteen tiedostojärjestelmään. [12.]

TC65 GSM -moduulia voidaan kontrolloida käyttämällä AT-komentoja. TC65 GSM -moduuli tarjoaa enimmillään kolme ilmentymää Siemensin kirjaston ATCommand-rajapintaa

toteuttavasta luokasta. SYSW-järjestelmäkirjasto on ottanut kaksi ATCommand-rajapintaa toteuttavan luokan ilmentymää käyttöön. Yksi on varattu SYSW-kirjaston palveluille ja toinen sarjaväylän (SPI) kommunikointiin apuprosessorin kanssa. Asiakassovellukselle jää yksi ilmentymä käytettäväksi ATCommand-rajapinnasta. Tämän kiertämiseksi SYSW-järjestelmäkirjastoon on kehitetty rajapinta IAtManager, joka toteuttaa synkronoidun kääreen (syncronized wrapper) viimeisen vapaana olevan ATCommand-rajapinnan ympärille. Rajapinta IAtManager mahdollistaa viimeisen ATCommand-rajapinnan jakamisen useiden eri palveluiden kesken. [12.]

Rajapinta ICopWatchdog tukee SYSW-järjestelmäkirjaston ja apuprosessorin kahdenkeskistä vahtikoiramekanismia. Vahtikoira on yleinen mekanismi, jonka avulla jumiin mennyt sovellus käynnistetään automaattisesti uudestaan. Mikäli jompikumpi, asiakassovellus tai apu-prosessori, lakkaa vastaamasta vahtikoiralle, laite käynnistetään uudestaan. [12.]

5.2 Ajoneuvotietokoneen sulautettu Java-ohjelmisto - FLSender

FLSender on oliopohjainen sulautettu Java imlet-sovellus, jota suoritetaan A1 Flex -ajoneuvotietokoneen GSM-moduulissa. Ajoneuvotietokoneen sovellus (FLSender) kerää ajoneuvosta tietoja ja lähettää ne pitempiaikaista säilytystä varten palvelimelle. FLSender -sovellus on itsenäinen Java-ohjelma, jonka suoritus käynnistyy automaattisesti ajoneuvon käynnistyessä ja sammuu hallitusti virran katketessa ajoneuvosta.

5.2.1 FLSender-sovelluksen vaatimukset

Ajoneuvotietokoneen ohjemiston pitää kyetä toimimaan itsenäisesti pitkään ilman käyttäjän toimenpiteitä. Tämä asettaa erityisiä vaatimuksia ohjelmiston toiminnalle ja vikatilanteiden hallinnalle. Ohjelmiston vaatimuksiin kuuluu mm.

- olla varmatoiminen
- toipua vikatilanteista
- olla konfiguroitavissa
- hallita tietojen puskurointi tietoliikenneverkon vikatilanteessa
- käynnistyä ja toimia itsenäisesti syöttövirran kytkeytyessä päälle
- lopettaa ohjelman suoritus hallitusti syöttövirran katkettua
- olla helposti laajennettavissa uusille laitteille.

Varmatoimisuus tarkoittaa sitä, että ohjelma on katkeamattomasti suoritettavissa Java-virtuaalikoneessa. Mahdollisia ongelmia voi tuottaa resurssien hallitsematon käyttö, kuten muistin täyttyminen. Vaikka Java-virtuaalikone huolehtii automaattisesti tuhottujen olioiden vapauttamisesta muistista (roskienkeruu), ohjelmoijan tulee silti ottaa huomioon ohjelman teknisessä toteutuksessa järkevä olioiden luonti. Jokainen tarpeeton olion luonti tai poistaminen muistista kuluttaa siivun prosessoriajasta. Toisaalta kaikkien olioiden luominen pysyvästi muistiin kuluttaa kohtuuttomasti muistia tai jopa täyttää sen kokonaan.

Vikatilanteista toipuminen on ohjelmiston kriittisimpiä osa-alueita. Ohjelmointikielenä Java tarjoaa vikatilanteiden hallintaan tehokkaat toiminnot. Kaikki arvelluttavat koodialueet voidaan suojata erillisillä vikatilanteiden hallintaan tarkoitetuilla koodilohkoilla. Mikäli vikatilanne laukeaa, se voidaan käsitellä hallitusti. Ohjelmiston tulee tuntea kaikki mahdolliset vikatilanteet ja hallita sopivat vaihtoehdot vikatilanteista toipumiseen.

Konfiguroitavuus tarkoittaa ohjelmiston sovellettavuutta monenlaisiin tarpeisiin. Ohjelmistoon ladataan tarpeelliset asetukset, jonka jälkeen se alkaa suorittaa konfiguraation mukaisia toimenpiteitä. Konfiguraatiolla otetaan käyttöön erilaisia seurattavia laitteita, määritetään laitteiden asetuksia ja määritetään sovelluksen yleisiä asetuksia. Laitteen konfiguroinnin suorittaminen pitää olla mahdollista tehdä etänä ilman, että laite on käsiteltävissä.

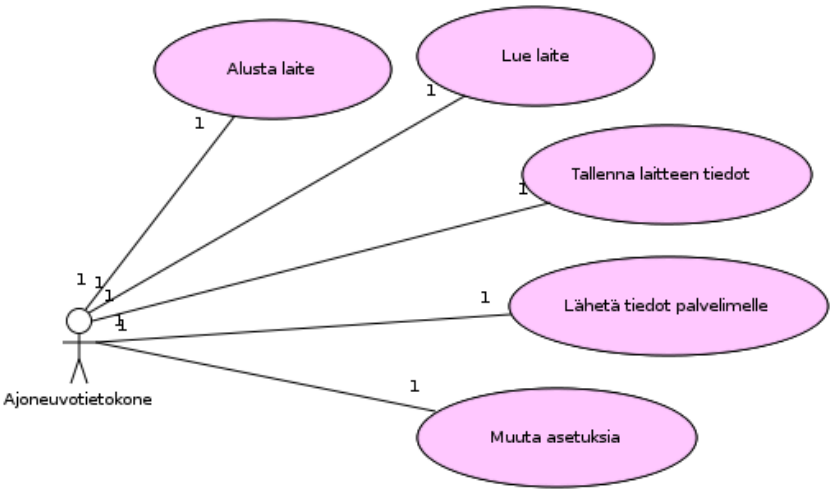
Tietoliikenneverkot voivat olla monista eri syistä osan aikaa tai toisinaan kokonaan tavoittamattomissa. Tietoliikenneverkon vikatilanteessa tai katvelalueella olevan ajoneuvon tiedot tulee puskuroida paikallisesti ja lähettää palvelimelle tietoliikenneyhteyksien palatessa. On tärkeää, että tietoja ei pääse katoamaan, vaan kaikki kerätyt tiedot voidaan tarvittaessa myöhemmin purkaa laitteesta.

Laitteen tulee käynnistyä automaattisesti ajoneuvon käynnistymisen yhteydessä. Samoin ajoneuvon sammuttaminen laukaisee toiminnon, joka sammuttaa ajoneuvotietokoneen automaattisesti määrätyn ajan jälkeen. Näin estetään laitteen turha päälläolo akun varassa ja pitkitetään akun käyttöikää.

Ajoneuvotietokoneen ohjelmiston tulee olla laajennettavissa helposti erilaisille uusille laitteille. Suunnitteluvaiheessa pitää soveltaa tunnettuja ohjelmistokehityksen suunnittelumalleja yleisimpien ongelmien ratkaisuissa. Se takaa paremmat mahdollisuudet sovelluksen tulevalle jatkokehitykselle ja ylläpidolle.

5.2.2 Käyttötapaukset

Kuvassa 19 on UML-käyttötapauskaavio, joka esitettää FLSender-ohjelmiston viisi oleellisinta käyttötapausta. FLSender-ohjelmiston käyttötapauksia ovat laitteiden alustaminen ja lukeminen, tietojen tallentaminen, tietojen lähetys palvelimelle ja ohjelman asetusten muuttaminen.



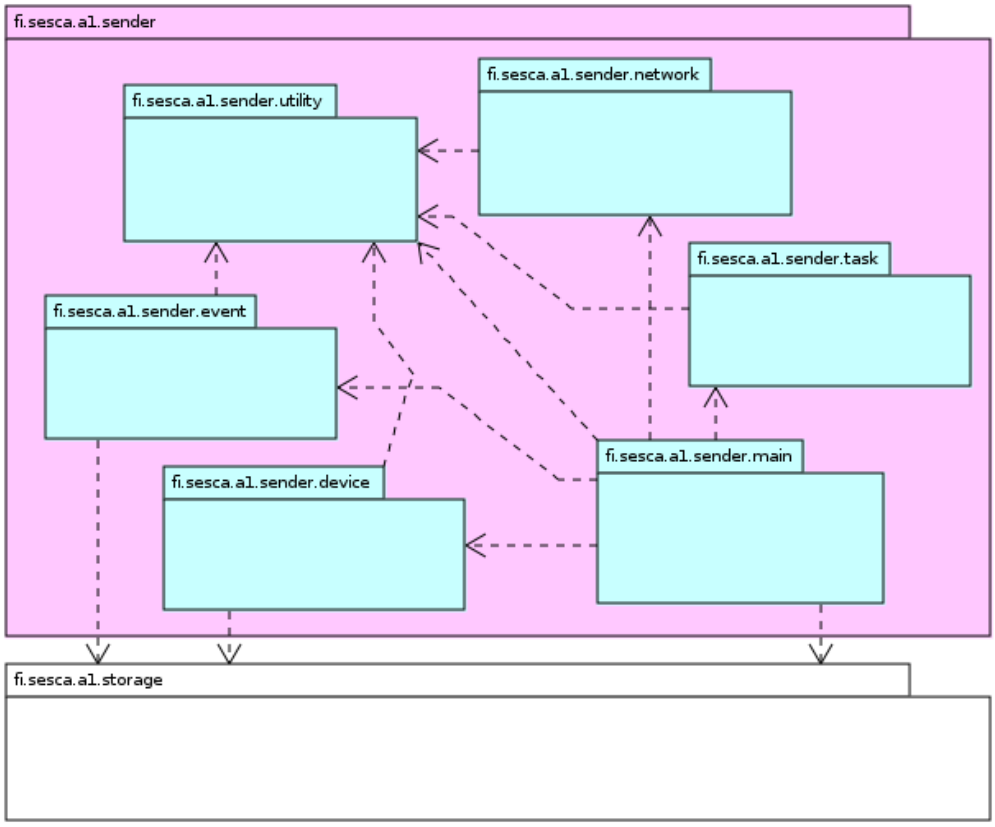
Kuva 19. Ajoneuvotietokoneen ohjelmiston käyttötapaukset

Kuvan 19 ajoneuvotietokoneen FLSender-ohjelmiston käyttötapauskaavio on suhteellisen yksinkertainen. Siinä on yksi käyttäjä (actor), joka toteuttaa viisi erilaista käyttötapausta (use case). Laitteen alustaminen tarkoittaa sopivien asetusparametrien lähettämistä ohjelmiston käynnistysvaiheessa laitteelle, jonka jälkeen laite toimii itsenäisesti ja on luettavissa. Ajoneuvotietokone lukee ja tallentaa laitteiden tiedot sekä lähettää ne palvelimelle. Ajoneuvotietokone osaa myös vastaanottaa uusia asetuksia ja ottaa ne käyttöön.

5.2.3 Luokat ja pakkaukset

Ajoneuvotietokoneen FLSender-ohjelmistosta on laadittu luokkakaavio (liite 1). FLSender-sovelluksen luokkakaavio sisältää noin 40 luokkaa, jotka on jaoteltu omiin loogisiin kokonaisuuksiin. Java mahdollistaa luokkakokonaisuuksien jaottelun pakkausten avulla. Pakkauksiin kootaan loogisesti yhteen kuuluvia luokkia omiksi kokonaisuuksiksi. Toimintojen ryhmittely pakkauksiin selkeyttää ohjelmistoa ja tekee siitä paremmin ylläpidettävää. Kuvan 20 UML-kaavio esittää pakkauksia, joista FLSender-sovellus koostuu sekä pakkausten välisiä riippuvuussuhteita. Sen pohjalta on helppo hahmottaa sovelluksen sisältämiä suurempia toimintakokonaisuuksia astetta korkeammalla tasolla. Alimmalla tasolla on kaksi pakkausta: fi.sesca.a1.sender ja fi.sesca.a1.storage. Pakkaus fi.sesca.a1.sender sisältää kaiken FLSender-sovelluksen koodin, joka on sisäisesti jaoteltu vielä omiin pienempiin pakkauksiin.

Pakkaus fi.sesca.a1.storage on FLSender- ja FLReceiver-sovelluksien käyttämä yhteinen tietovarasto. FLSender-sovellus käyttää tietovarastopakkausta laitteilta luettujen tietojen väliaikaiseen varastointiin ajoneuvotietokoneessa ja FLReceiver-sovellus XML-viestien sisältämien tietojen väliaikaiseen varastoimiseen palvelimella.



Kuva 20. FLSender-sovelluksen UML-pakkauskaavio esittää pakkaukset ja niiden väliset riippuvuudet

Kuvassa 20 FLSender-sovelluksen sisältämä pääpakkaus fi.sesca.a1.sender.main käyttää muita pakkauksia toteuttaakseen ohjelmiston toiminnan. Pääpakkaus fi.sesca.a1.sender.main sisältää FLSender- ja MessageSender-säieluokat. FLSender-säieluokka on koko sovelluksen pääluokka, josta sovelluksen suoritus käynnistetään. FLSender-luokka käyttää esimerkiksi fi.sesca.a1.sender.device-pakkausta lukeakseen ajoneuvon laitteiden tiedot ja fi.sesca.a1.sender.utility-pakkausta tallentaakseen tiedot FLASH-levylle. MessageSender-luokka vastaa itsenäisesti tiedostojen lähettämistä palvelimelle. MessageSender käyttää esimerkiksi fi.sesca.a1.sender.utility-pakkauksen tarjoamia tiedostonkäsittelypalveluja lukekaseen lähetettävät tiedostot FLASH-levyltä.

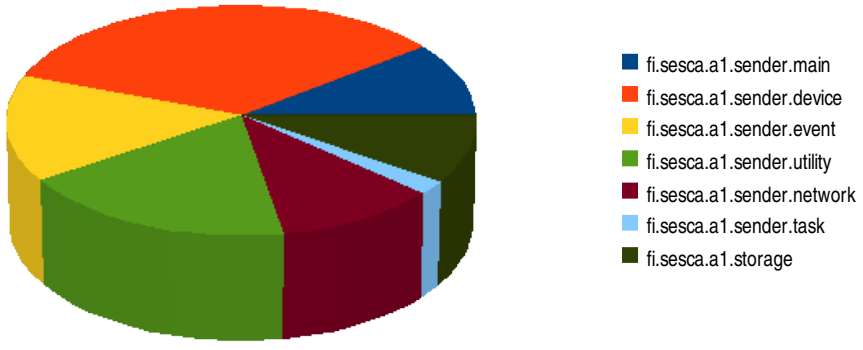
Pakkaukseen fi.sesca.a1.sender.device on ohjelmoitu suurin osa laitteiden käsittelemiseen tarvittavista rutiineista. Nämä rutiinit on ohjelmoitu rekursiokoostemallia hyväksi käyttäen. Osa laitteiden lähettämistä tiedoista saadaan tapahtumina, ja pakkaus fi.sesca.a1.sender.event

sisältää rutiinit tapahtumien käsittelemistä varten. Järjestelmässä ilmenevä tapahtuma käsitellään kullekin tapahtumalla ohjelmoidulla käsittelijällä, joka on toteutettu säikeenä. Pakkaus fi.sesca.a1.sender.utility sisältää kaikki yleiset sovelluksen tarvitsemat apuluokat. Niitä ovat esimerkiksi tiedostojenkäsittely- ja konfiguraatioluokat. Pakkaus fi.sesca.a1. sender.network sisältää tekstiviesti- ja GPRS-yhteyksiin tarvittavat rutiinit. Sitä käytetään esimerkiksi etäkonfiguroinnin toteutukseen ja tiedostojen lähettämiseen palvelimelle. Pakkaus fi.sesca.a1.sender.task sisältää ajastintoimintoja sisältävät luokat. Tällainen on esimerkiksi ajoneuvotietokoneen sammutuksesta vastaava ShutDownTimerTask-luokka, joka sammuttaa hallitusti sovelluksen asetuksissa määritellyn ajan kuluttua ajoneuvon sammuttamisesta.

Taulukko 1. FLSender-sovelluksen v0.7 sisältämät pakkaukset ja niiden koot

Pakkauksen nimi	**Pakkauksen kuvaus**	**Koko (koodiriviä)**	**Suhteellinen koko (%)**
fi.sesca.a1.sender	Alatason pakkaus, joka koostuu pienemmistä pakkauksista	-	-
fi.sesca.a1.sender.main	FLSender-sovelluksen pääpakkaus	678	10,3
fi.sesca.a1.sender.device	Laitteet toteuttava pakkaus	2245	34,1
fi.sesca.a1.sender.event	Tapahtumien hallinnan toteuttava pakkaus	982	14,9
fi.sesca.a1.sender.utility	Sovelluksen aputoiminnot sisältävä pakkaus	1206	18,3
fi.sesca.a1.sender.network	Verkkoyhteydet toteuttava pakkaus	709	10,8
fi.sesca.a1.sender.task	Ajastetut tehtävät sisältävä pakkaus	130	2,0
fi.sesca.a1.storage	Tietovarastopakkaus FLSender- ja FLReceiver-sovelluksille	626	9,5

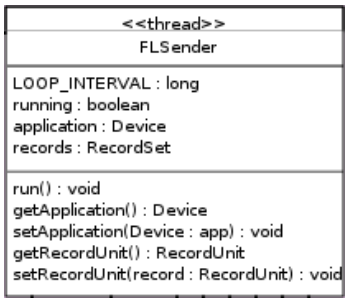
Kuvasta 21 nähdään FLSender-ohjelmiston pakkausten suhteellinen koko graafisesti. Kaaviosta huomioidaan, että kolme suurinta pakkausta ovat fi.sesca.a1.sender.device, fi.sesca.a1.sender.utility ja fi.sesca.a1.sender.event. FLSender-ohjelmiston kaikkien pakkausten yhteenlaskettu koodirivimäärä on 6370. Liitteessä 5 (CD) on pakkausten Java API.



Kuva 21. FLSender-sovelluksen v0.7 pakkausten suhteelliset koot

5.2.4 FLSender-pääluokka

Pakkaus fi.sesca.a1.sender.main sisältää FLSender-luokan, joka on ajoneuvosovelluksen ydinluokka. Kuvan 22 luokkakaavioon on merkitty vain tärkeimmät asiat. Tosiasiassa FLSender-luokka koostuu monista muistakin luokista, jotka hallitsevat puolestaan omia kokonaisuuksia. FLSender-luokka perii Java ME Midlet -luokan ominaisuudet.

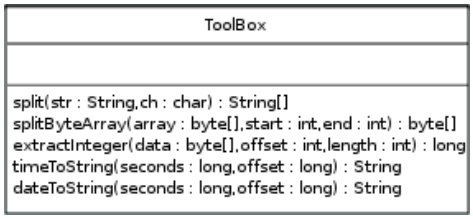


Kuva 22. FLSender-sovelluksen pääluokka

Käyttöjännitteen kytkeydettyä ajoneuvotietokone alkaa suorittaa FLSender-luokan koodia. Luokka tekee monenlaisia alustuksia. Ensin se alustaa yhteydet apuprosessorille luomalla ilmentymät tärkeimmistä olioista (Sysw, ICopDriver, IAtManager). Tämän jälkeen se luo ilmentymät vähemmän tärkeimmistä olioista (Settings, FileHandler, GeneralEventListener, NetMonitor, SmsTextConnection, MessageSender, ApplicationBuilder). Kun sovelluksen toiminnan kannalta kaikki tarvittavat ilmentymät on luotu ja alustettu, luodaan ja alustetaan varsinainen Device-olio, joka hallitsee ajoneuvon laitteita. FLSender alkaa tästä eteenpäin pyöriä ikuisessa silmukassa, jonka päätteeksi se nukkuu jonkin aikaa. Silmukassa pyydetään Device-oliota lukemaan laitteilta tiedot ja varastoimaan ne RAM-muistiin. Silmukan lopussa tiedot siirretään RAM-muistista ajoneuvotietokoneen FLASH-levylle.

5.2.5 ToolBox-apuluokka

FLSender-sovelluksen jokainen luokka on laadittu tarkkaan tekemään tiettyä tehtävää. Osa sovelluksen luokista tuntee toisensa, ja luokkien keskinäinen vuorovaikutus tuottaa lopulta halutun toiminnallisuuden. Kuvassa 23 on esitetty yksi pakkauksen fi.sesca.a1.sender.utility -luokista, jonka tehtävänä on olla yleinen työkalupakki. Sieltä koko muu sovellus voi käyttää tarvittuja työkaluja erilaisten tehtävien toteuttamiseen. ToolBox-luokka sisältää enimmäkseen sellaisia apufunktioita, joita Java ME:n kirjastoista ei vakiona löydy. Siksi se on jouduttu toteuttamaan itse ohjelmoimalla.



Kuva 23. ToolBox-luokka sisältää yleisiä työkaluja koko sovelluksen käyttöön

ToolBox-luokan split-funktio palauttaa paloitellun merkkijonon taulukossa. Se paloittelee parametrina annetun merkkijonon osiin annetusta erotinmerkistä. Ensimmäisenä parametrina on merkkijono, ja toinen parametri on char-tyyppinen erotinmerkki. FLSender-sovelluksessa funktiota käytetään esimerkiksi tekstiviestillä saapuneiden konfigurointiasetusten parsimiseen.

Funktio splitByteArray ottaa parametrinään byte-tyyppisen taulukon ja kaksi kokonaislukua. Funktio mahdollistaa byte-taulukon paloittelemisen halutusta aloituskohdasta haluttuun lopetuskohtaan. Toinen parametri on paloittelun aloituskohta ja kolmas parametri paloittelun lopetuskohta. Funktio palauttaa aloitus- ja lopetuskohtien määrittämän alueen byte-taulukkona. Funktiota käytetään esimerkiksi CAN-väylätietojen erotteluun. FLSender-sovelluksessa monia CAN-väylätietoja saadaan usein yhdessä isossa byte-taulukossa ja tiedot halutaan erotella omiin taulukoihin pilkkomalla iso taulukko sopivista kohdista pienempiin taulukoihin.

Java-ohjelmointikielessä ei ole valmista toteutusta byte-tyyppisen taulukon konvertointiin kokonaisluvuksi. Siihen tarkoitukseen on ohjelmoitu oma funktio extractInteger. Koodilistaus 3 esittää funktion toteutuksen. Se ottaa ensimmäisenä parametrinaan byte-taulukon, josta muodostetaan kokonaisluku. Toinen parametri on offset, jolla voidaan määrätä aloituskohta byte-taulukosta muodostettavalle kokonaisluvulle. Kolmas parametri on length, joka kertoo byte-taulukon pituuden. Funktiota käytetään esimerkiksi byte-taulukkona saatujen CAN-väylätietojen konvertointiin kokonaisluvuiksi.

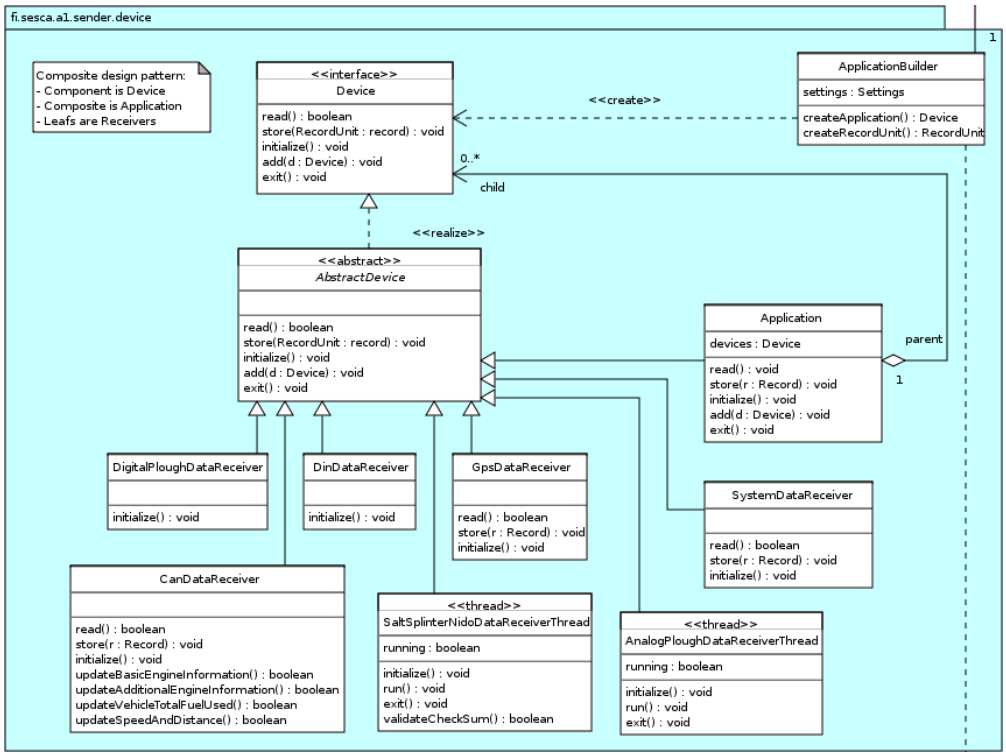
Koodilistaus 3. ExtractInteger-funktion toteutus

```
1  public static long extractInteger(byte[] data, int offset, int length) {
2      if(offset + length > data.length) {
3          return 0;
4      }
5
6      long result = 0;
7      for(int i=0; i<length; i++) {
8          result = (result << 8) + (data[offset+i] & 0xff);
9      }
10     return result;
11 }
```

Funktiot timeToString ja dateToString ottavat kaksi kokonaislukuparametriä. Ensimmäinen kokonaisluku kertoo ajan millisekunteina. Toinen parametri on aikaan lisättävä offset. Offsetillä voidaan toteuttaa kätevästi esimerkiksi eri aikavyöhykkeen ajat. Funktio palauttaa ajan ja päivämäärän esityksen String-muotoisena. Aika esitetään muodossa hh:mm:ss, jossa hh on tunnit, mm on minuutit ja ss on sekunnit. Päivämäärä esitetään muodossa yyyy-mm-dd, jossa yyyy on vuodet, mm on kuukaudet ja dd on päivät. Java ME -ympäristössä virtuaalikoneelta saatava reaaliaikakellon aika tulee millisekunteina, ja funktioilla muunnetaan aikaleima valmiiksi tietokannan ymmärtämään datetime-muotoon.

5.2.6 Ajoneuvon laitteiden toteutukset

Kuvassa 24 on FLSender-sovelluksen pakkaus fi.sesca.a1.sender.device. Pakkaus pitää sisällään suurimman osan laitteiden toteutuksesta. Laitteet on toteutettu rekursiokooste-suunnittelumallilla. Kukin laite suorittaa saman tyyppisiä operaatioita, mutta kuitenkin niin, että kaikille laitteille operaatiot ohjelmoidaan eri tavalla - laitteen vaatimalla tavalla. FLSender-sovelluksessa toteutettuja laitteita ovat järjestelmälaite (SystemDataReceiver) kellonajan ja imei-koodin selvittämiseen, GPS-laite (GpsDataReceiver) paikkatiedon vastaanottamiseen, CAN-laite (CanDataReceiver) ajoneuvon moottorin tietojen lukemiseen, auralaitteet (DigitalPloughDataReceiver, AnalogPloughDataReceiverThread) aurojen tarkkailuun, suolasirotinlaite (SaltSplinterNidoDataReceiverThread) suolasirottimen tietojen lukemiseen sekä digitaalisen signaalin havainnoiva laite (DinDataReceiver). Laitteen alustaminen (initialize) tapahtuu laitekohtaisesti, jossa kullekin laitteelle lähetetään sen toiminnan kannalta tärkeät asetukset. Alustuksen jälkeen käyttövalmiilta laiteelta luetaan (read) tiedot joko määräajoin tai saadaan ne erillisen tapahtuman kautta. Laitteen lukemisen jälkeen tiedot varastoidaan (store). Laitteen toiminta pitää tarvittaessa pystyä myös pysäyttämään (exit), jos kyseessä on säie-tyyppinen laite.



Kuva 24. Rekursiokoostesuunnittelumallin hyödyntäminen laitteiden toteutuksessa

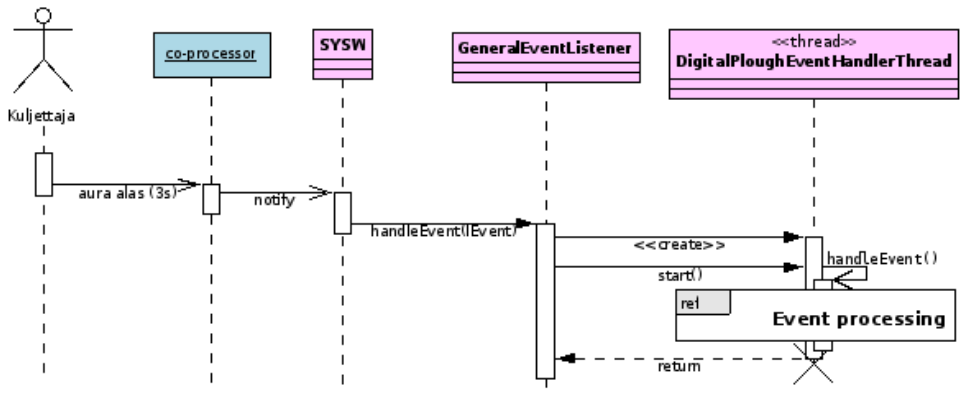
Kaikkia laitteita käsitellään samannimisillä metodeilla, jotka on määritelty Device-rajapinnassa. Device-rajapinnan toteuttaa AbstractDevice-luokka, jonka päätarkoitus on olla väliluokkana rekursiokoostemallin lehdille (laitteille). Lehdet ovat luokkia, jotka muodostavat jonkin laitteen konkreettisen toteutuksen. AbstractDevice-väliluokka ei pakota ohjelmoimaan kaikkia Device-rajapinnassa määriteltyjä metodeja lehdissä, mutta tarvittaessa ne voidaan toteuttaa. Perinteisessä Erich Gamman mukaisessa rekursiokoostesuunnittelumallissa ei AbstractDevice-väliluokkaa ole, mutta siihen on päädytty, koska useissa lehdissä käytetään vain osaa Device-rajapinnan metodeista. Jos AbstractDevice-väliluokkaa ei olisi, jouduttaisiin moniin lehtiin ohjelmoimaan käyttämättömömien metodien tyhjät rungot (stub method).

Kaikki lehdet perivät AbstractDevice-luokan ja antavat toteutuksen laitteen toiminnan kannalta oleellisille rajapinnan metodeille. Koosteluokka Application toteuttaa Device-rajapintaa ja sisältää taulukon, johon lehdet (laitteet) lisätään. Koosteluokka Application käsittelee lehtiä Device-rajapinnan määrittämillä metodeilla. Kun Application-oliota pyydetään esimerkiksi alustumaan, se ajaa silmukassa alustuskutsun yksitellen kaikille taulukossa oleville lehdille. Luokka Application antaa myös metodille add konkreettisen toteutuksen, jolla lisätään lehtiä luokassa olevaan taulukkoon. Luokka ApplicationBuilder on rakentajaluokka, joka rakentaa sovelluksen konfiguraatioon perustuvan Device-rajapinnan mukaisen Application-olion. ApplicationBuilder käyttää Application-olion add-metodia ja lisää lehden taulukkoon, mikäli

lehti on konfiguraatiossa otettu käyttöön. Rakentajaluokalla saadaan helposti luotua uusi Application-olio esimerkiksi konfiguraation muuttumisen jälkeen. Rakentajaluokka osaa myös luoda konfiguraatioon perustuvan tietovaraston Application-olion suorittamaa tietojen varastointia (store) varten.

5.2.7 Tapahtumien käsittely

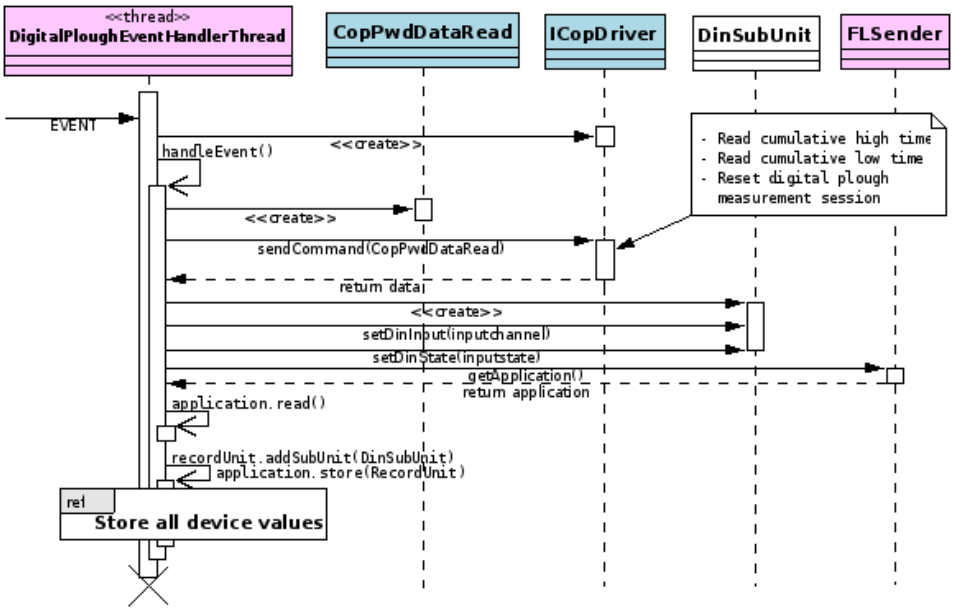
FLSender-sovelluksen versiossa 0.7 saadaan digitaalisten aurojen ja digitaalisten sisääntulojen tilatiedot sekä tekstiviestien saapuminen tapahtumina. Kuvan 25 sekvenssikaavio havainnollistaa tapahtumaketjua alkupäästä loppupäähän yleisellä tasolla käyttäen esimerkkinä auraustapahtumaa. Kuvan 25 tilanteessa edellisen luvun digitaalinen auralaite DigitalPloughDataReceiver on aiemmin alustanut apuprosessorin tarkkailemaan tietyn mittaista digitaalista kumulatiivista signaalin päälläoloaikaa niissä IO-väylissä, joihin aurat on kytketty. Kuljettajan ajaessa etuaruraa 3 sekuntia alas (ylitse konfiguraatiossa määritetyn raja-arvon) apuprosessori (co-prosessor) ilmoittaa SYSW-järjestelmäkirjastolle tapahtumasta. Järjestelmäkirjasto jakelee tapahtuman kuvassa olevalle GeneralEventListener-luokalle. Se tutkii tapahtuman tyypin, luo sille sopivan käsittelijäluokan (DigitalPloughEventHandlerThread) ja käynnistää käsittelijän kutsumalla säikeen käynnistysmetodia start. Auraustapahtumien käsittelijä (DigitalPloughEventHandlerThread) käsittelee etuauran ja alusterän tuottamat tapahtumat. Tapahtuman käsittelijäluokka on aina säie, joka kuolee suorituksensa jälkeen.



Kuva 25. UML-sekvenssikaavio etuauran tai alusterän alhaalle ajamisesta

Kuvassa 26 on esitetty tarkemmin kuvaan 25 liittyvän DigitalPloughEventHandlerThread-säieluokan käsittelemän auraustapahtuman prosessointia (event processing). GeneralEventListener-luokan delegoidessa tapahtuman DigitalPloughEventHandlerThread-luokalle se hankkii ensin itselleen viitteen ICopDriver-oliosta, jota se tarvitsee kommunikoidakseen apuprosessorin kanssa. Tämän jälkeen se luo ja lähettää

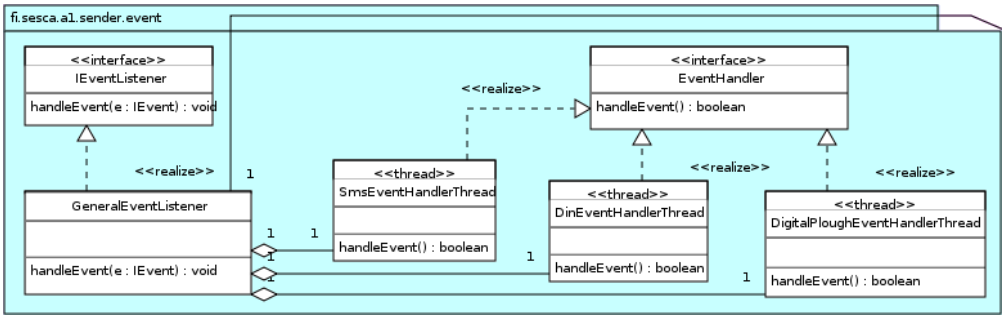
apuprosessorille CopPwdDataRead-olion, jonka avulla luetaan kumulatiivinen signaalin ylhäälläolemisaika, kumulatiivinen signaalin alhaallaolemisaika sekä resetoidaan mittaustapahtuma. Mittaustapahtuman resetointi käynnistää automaattisesti uuden mittaustapahtuman seurannan apuprosessorilla. Kun tiedot on luettu apuprosessorilta, auratapahtumien käsittelijäluokka luo tietovarasto-olion (DinSubUnit) ja asettaa sinne signaalin tilan sekä I/O-kanavan numeron, josta tapahtuma saatiin. Tietojen tallentamisen jälkeen pyydetään FLSender-oliolta viite koko sovelluksen ylimpään laiteolioon ja pyydetään laiteoliota päivittämään kaikkien laitteiden viimeisimmät tiedot laitteilta. Tällä menettelyllä liitetään auraustapahtumaan esimerkiksi tarkat GPS- ja CAN-tiedot. Viimeisenä isona tapahtumaryppäänä viedään kaikki oliotietorakenteeseen tallennetut tiedot talteen XML-viestiksi FLASH-levylle. Tämä tapahtumaprosessi on kuvattu sivun 42 kuvassa 28.



Kuva 26. Auaraustapahtumakäsittelijän toiminta tapahtuman saapuessa

Kuvan 27 pakkaukseen fi.sesca.a1.sender.event on koottu kaikki tapahtumien käsittelyyn osallistuvat luokat. Aplicomin SYSW-järjestelmäkirjasto mahdollistaa tapahtumien käsittelyn tarkkailijasuunnittelumalliin (observer design pattern) perustuen. Tapahtumien kuuntelua varten ohjelmoidaan luokka GeneralEventListener, joka toteuttaa SYSW-järjestelmäkirjaston IEventListener-rajapinnan mukaisen handleEvent-metodin. GeneralEventListener-olio on rekisteröity sovelluksen alustusrutiineissa järjestelmäkirjastolle, joten se on valmis vastaanottamaan järjestelmäkirjastolta pyyntöjä tapahtumien käsittelyistä. Luokan GeneralEventListener handleEvent-metodissa tutkitaan tapahtuman tyyppi ja ohjataan se suoraan eteenpäin sitä varten ohjelmoidulle käsittelijäsäikeelle. Käsittelijät ovat EventHandler-rajapinnan toteuttavia säieluokkia. Digitaalisten aurojen tapahtumien

käsittelemiseen on ohjelmoitu luokka DigitalPloughDataReceiverThread. Digitaalisten sisääntulojen tapahtumiin on ohjelmoitu vastaavasti luokka DinEventHandlerThread ja tekstiviestien käsittelemiseen on ohjelmoitu luokka SmsEventHandlerThread.



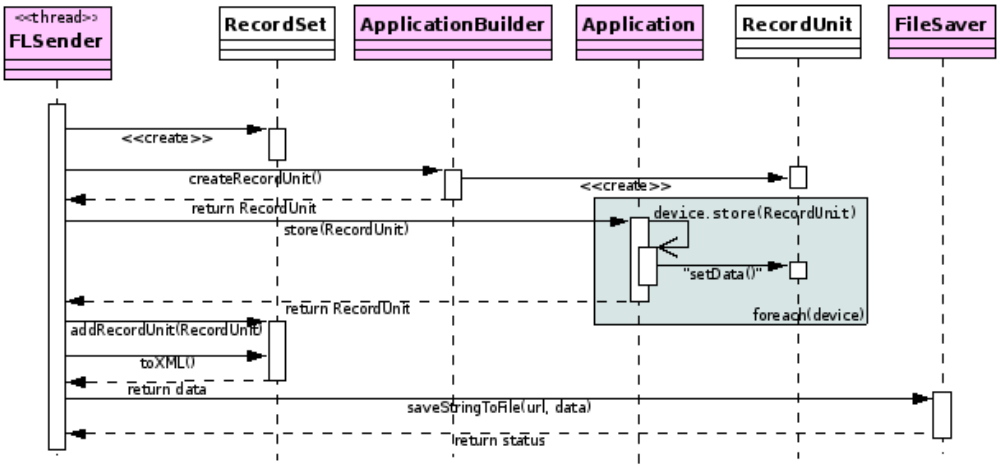
Kuva 27. Pakkaus fi.sesca.a1.sender.event hallitsee tapahtumienkäsittelyt

Kaikki tapahtumienkäsittelijät on toteutettu säikeinä. Esimerkiksi tekstiviestin vastaanottaminen ja prosessointi omassa säikeessä mahdollistaa samanaikaisen GPRS-yhteyden palvelimelle. Yleisesti ottaen säikeiden ohjelmointia pidetään haastavana, koska tällöin järjestelmän testaaminen muuttuu astetta hankalammaksi. Testaamisen hankaloituminen johtuu siitä, että koskaan ei tiedetä missä järjestyksessä kaikki säikeet tulevat suoritetuksi. FLSender-sovelluksessa säikeiden ohjelmointi on kuitenkin melko suoraviivaista, koska säikeet eivät jaa keskenään kriittisiä tietoja, jotka tulisi suojata toisiltaan.

5.2.8 Ajoneuvosta kerättyjen tietojen tallentaminen

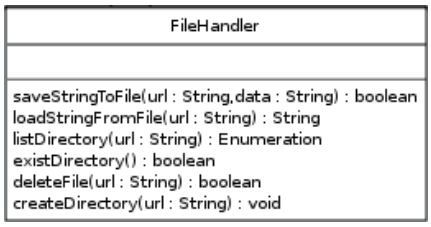
FLSender-sovelluksessa laitteilta kerätyt tiedot varastoidaan ohjelman suoritusvaiheessa väliaikaisesti RAM-muistiin. Kun kaikki laitteiden tiedot on saatu koottua RAM-muistiin, tiedot tallennetaan laitteen pysyvään FLASH-muistiin XML-muotoiseksi tekstitiedostoksi (liite 2). Pakkaus fi.sesca.a1.storage (liite 1) koostuu RAM-muistissa pidettävästä oliojoukosta, johon laitteiden tiedot tallennetaan väliaikaisesti. Kuvassa 28 FLSender-pääluokka aloittaa tietojen varastoinnin luomalla ensin RecordSet-tyyppisen tietovarasto-olion. RecordSet-olio on tietovaraston ylin olio, joka voi tarvittaessa sisältää useita RecordUnit-olioita (useita peräkkäisiä mittauksia). FLSender-luo ApplicationBuilder-oliota apuna käyttäen konfiguraatioon pohjautuvan RecordUnit-olion. Koska RecordUnit-olio pohjautuu konfiguraatioon se sisältää vain käytössä oleville laitteille tarvittavat tietovarastoyksiköt (CanSubUnit, GpsSubUnit, jne.). Tämän jälkeen FLSender pyytää laitteiden ylintä Application-oliota varastoimaan kaikkien käytössä olevien laitteiden tiedot annettuun RecordUnit-tietovarasto-olioon. Käytännössä se on silmukka, jossa pyydetään kutakin laiteoliota varastoimaan tiedot omiin pienempiin tietovarastoyksikköihin, jotka on liitetty

RecordUnit-olioon. Kun tiedot on laitekohtaisesti varastoitu RecordUnit-olioon FLSender lisää tämän RecordSet-olioon. Tämän jälkeen FLSender pyytää RecordSetiltä koko oliojoukon tiedot XML-muodossa. Lopuksi FLSender lähettää RecordSetiltä saamansa XML-datan FileSaver-oliolle. FileSaver-olio tallentaa XML-viestin FLASH-tiedostojärjestelmään omaksi tiedostoksi ja paluttaa onnistumisesta totuusarvon.



Kuva 28. Sekvenssikaavio tietojen tallentamisesta laitteilta FLASH-levylle

Kuvan 29 luokka FileHandler sijaitsee pakkauksessa fi.sesca.a1.sender.utility. Luokka on yleinen tiedostojen käsittelyyn ohjelmoitu luokka. Sitä käytetään esimerkiksi tietovarasto-oliosta saadun XML-viestin tallentamiseen FLASH-tiedostojärjestelmään.

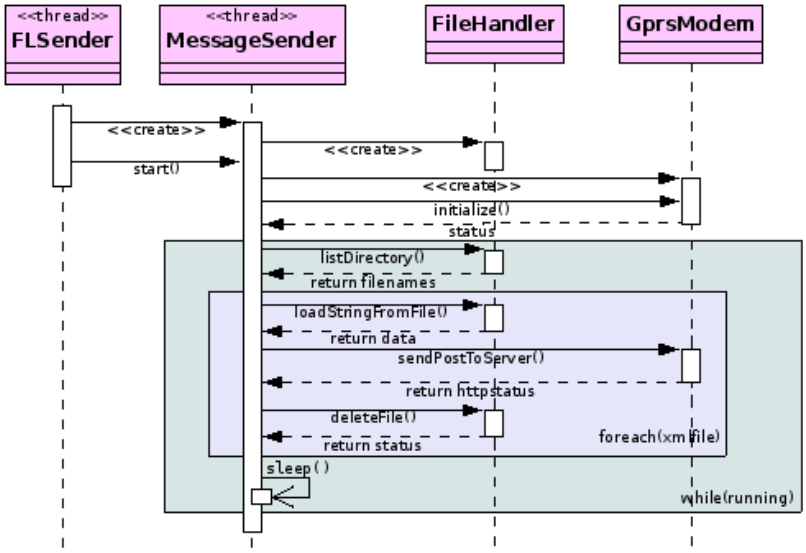


Kuva 29. FileHandler-luokka mahdollistaa tiedostojenkäsittelyn FLASH-levyllä

FileHandler-luokan saveStringToFile-metodi saa ensimmäisenä parametrina polun, johon tiedosto halutaan tallentaa ja toisena parametrina tallennettavat tiedot merkkijonotyyppisessä muuttujassa. Metodi palauttaa totuusarvon tietojen tallentamisen onnistumisesta. Luokka sisältää myös muita yleisiä metodeja tiedostojen hallintaan. Metodilla loadStringFromFile saadaan FLASH-levyltä ladattua merkkijonotiedosto ohjelmaan. Metodi listDirectory antaa tiedostolistauksen halutusta hakemistosta ja metodilla createDirectory luodaan uusi hakemisto. Tiedostoja voidaan poistaa metodilla deleteFile FLASH-levyltä.

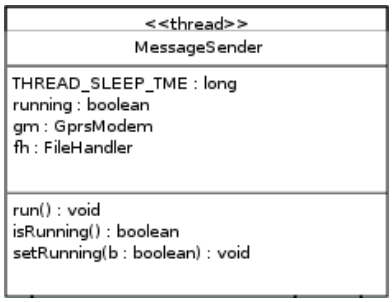
5.2.9 *Kerättyjen tietojen lähetys palvelimelle*

FLSender-sovelluksessa tietojen lähettämistä palvelimelle hallitsee säieluokka MessageSender. Kuvassa 30 FLSender-pääluokka luo alustusrutiineissa MessageSender-säikeen sekä pyytää säiettä käynnistymään. MessageSender luo ensimmäiseksi tiedostonkäsittelijäluokan (FileHandler) ja GPRS-modeemiluokan (GprsModem), jolle se lähettää alustuspyynnön. Tämän jälkeen MessageSender alkaa pyörimään ikuisessa silmukassa. MessageSender lukee FileHandler-olion avulla ensimmäisenä silmukan operaationa tiedostolistauksen lähetettävistä XML-viesteistä. Tämän jälkeen MessageSender menee sisempään silmukkaan, jossa se taas lukee FileHandler-oliota apuna käyttäen FLASH-levyltä ensimmäisen tiedostolistauksessa saaduista XML-viesteistä. MessageSender lähettää GprsModem-olion avulla luetun XML-viestin HTTP POST -kyselyllä palvelimelle. Jos viesti lähetettiin onnistuneesti palvelimelle (HTTP 200) MessageSender käyttää jälleen FileHandler-olion palveluja vapauttaakseen levytilaa poistamalla lähetetyn XML-viestin FLASH-levyltä. Kun kaikki sisemmän silmukan tiedostolistauksessa saadut XML-viestit on käsitelty, MessageSender poistuu sisemmästä silmukasta ja menee nukkumaan konfiguraation määrittämäksi ajaksi. Herättyään se lukee taas tiedostolistauksen ja toistaa samaa rutiinia (silmukoita).



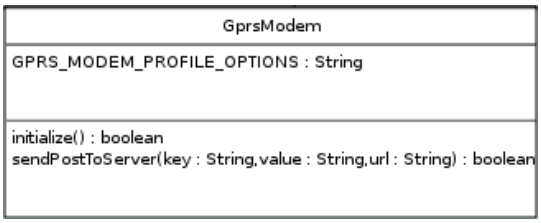
Kuva 30. Sekvenssikaavio MessageSender-olion tekemistä operaatioista lähettäessään tiedostot palvelimelle

Kuvassa 31 on MessageSender-luokka, joka koostuu GprsModem- ja FileHandler-luokista. FileHandler-luokka luodaan MessageSender-olion rakentajassa ja GprsModem-luokka säikeen alustusrutiinissa. Muuttujaan THREAD_SLEEP_TIME haetaan konfiguraatiosta säikelle nukkumisaika. Totuuasrvomuuttujan running avulla voidaan vapauttaa säie ikuisesta silmukasta vaihtamalla muuttujan tilaksi epätosi.



Kuva 31. MessageSender-säie vastaa tiedostojen lähettämisestä palvelimelle

Kuvan 32 GprsModem-luokalla on jäsenmuuttuja GPRS_MODEM_PROFILE_OPTIONS, joka sisältää konfiguraatiosta ladatut modeemin GPRS-asetukset. GPRS-asetukset sisältävät tyypillisesti GPRS-verkko-operaattorikohtaisen access point name -määrityksen. Access point name määrittelee operaattorin Internet-verkon, johon mobiililaite haluaa kytkeytyä. Lisäksi se määrittelee verkkoyhteyteen liitettävät operaattorin asetukset sekä valitsee sopivan asetusten joukon mobiililaitteen verkkoyhteydelle.



Kuva 32. GprsModem-luokalla lähetetään XML-viesti palvelimelle

GprsModem-luokassa on kaksi metodia. Metodissa initialize lähetetään GPRS-modeemille asetuksia ja tarkistetaan verkon tila. Metodilla sendPostToServer lähetetään HTTP POST -kysely palvelimelle. HTTP POST -kysely sisältää laitteilta kerätyt tiedot XML-muotoisena viestinä. HTTP POST -kysely sopii hyvin tietojen lähettämiseen palvelimelle, koska sillä saadaan esimerkiksi tieto palvelimelle lähetetyn kyselyn onnistumisesta (HTTP 200).

GprsModem-luokan tärkein metodi on sendPostToServer, joka muodostaa palvelimelle lähetettävän HTTP-kyselyn. HTTP-kysely muodostuu avain-arvo pareista, joissa kuhunkin avaimeen liitetään palvelimelle lähetettävä data. FLSenderin version 0.7 tekemässä HTTP

POST -kyselyssä voi esiintyä yksi avain, joka on jokin taulukossa 2 määritellyistä avaimista. Liitteessä 2 on version 3 mukainen XML-viesti, joka sisältää mm. järjestelmä, GPS, CAN sekä digitaalitietojen välittämiseen määritellyt tietorakenteet. Palvelin tunnistaa HTTP POST -kyselyn avaimen perusteella saapuneen viestin version ja osaa siten valita viestille sopivan käsittelijän. Tällaisella menettelyllä ei tarvitse sitoa XML-viestin rakennetta kiinteästi ja siten säilytetään mahdollisuus jatkossa liittää uudenlainen XML-viesti uuteen avaimeen, jolle myös palvelinpäässä ohjelmoidaan tarkoituksenmukainen käsittelijä.

Taulukko 2. FLSender v0.7 -ohjelmiston käyttämät HTTP POST -kyselyn avaimet, joihin liitetään tietyn version mukainen XML-sanoma

Avain	**Arvo**
XML-FLDataBox-Message-v1	Version 1 mukainen XML-viesti
FLSender-message-v2	Version 2 mukainen XML-viesti
FLSender-message-v3	Version 3 mukainen XML-viesti

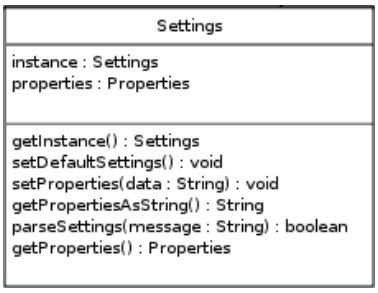
XML-muotoinen tietojen välitystapa palvelimelle ei sido palvelinpään sovellusta mihinkään tiettyyn ohjelmointikieleen. XML-muotoinen tietojen kuvaaminen takaa avoimemman liityntärajapinnan palvelimeen, joka on yksinkertainen toteuttaa erilaisissa tietojärjestelmissä. Se mahdollistaa joustavamman järjestelmän jatkokehityksen.

5.2.10 Sovelluksen konfiguroitavuus

FLSender-komponentti on suunniteltu siten, että se voidaan määritellä erillisen konfigurointitiedoston (liite 4) avulla toimimaan erilaisia laitteita sisältävissä ajoneuvoissa. Konfiguraatiotiedostossa määritetään sopivat parametrit esimerkiksi tienhoitoajoneuvoon, maankuljetusajoneuvoon ja nosturiautoon. Näin saadaan kulloinkin tarvitut tiedot kerättyä ja siirrettyä palvelimelle. Konfiguraatio määrittelee myös monenlaisia sovelluksen järjestelmäparametrejä, kuten tietojen lukemis- ja lähettämisaikavälin palvelimelle.

Sovellus tukee kahdenlaista konfigurointimenetelmää. Joskus on tarve päästä muuttamaan joitakin yksittäisiä parametrejä ja se onnistuu helposti SMS-pohjaisella etäkonfiguroinnilla. Sovellukselle voidaan lähettää tekstiviesti, joka sisältää konfigurointiasetuksia. Toinen mahdollisuus on muuttaa sovelluksen konfigurointitiedostoa (liite 4) ja kopioida se ajoneuvotietokoneen FLASH-levylle. Jälkimmäinen tapa vaatii erillisen ohjelmiston PC-ympäristöön, jolla konfiguraatiotiedosto kopioidaan FLASH-levylle.

Kuvan 33 Settings-luokka toteuttaa sovelluksen konfigurointiominaisuuden. Siinä on hyödynnetty ainokainen-suunnittelumallia (singleton). Settings-luokka on apuluokka, joka sisältää kaikki sovelluksen tarvitsemat asetukset, joiden pitää olla muutettavissa. Ainokainen-suunnittelumalli mahdollistaa vain yhden asetusluokan ilmentymän luonnin, jolla varmistutaan siitä, että muutettu asetus on saman tien kaikkialla sovelluksessa saatavilla. Asetusluokka sisältää yli 40 erilaista sovelluksen käyttämää parametriä.



Kuva 33. Konfiguroinnin toteuttaa Settings-luokka ainokainen-suunnittelumallilla

Settings-luokan ilmentymään muut luokat pääsevät käsiksi kutsumalla luokan staattista metodia getInstance. Settings-luokka sisältää Properties-olion, jossa ylläpidetään yksittäisiä asetuksia. Settings-luokka sisältää myös muutamia apumetodeja Properties-olion ylläpitämiseen ja rakentamiseen. Metodi getPropertiesAsString palauttaa Properties-olion sisältämät asetukset koottuna merkkijonoksi avain-arvo pareina. Sitä käytetään esimerkiksi tekstiviestillä muutettujen konfigurointiparametrien tallentamiseen FLASH-levyllä olevaan konfigurointitiedostoon. Näin vältetään muutettujen asetusten katoaminen, jos ajoneuvotietokoneesta katkeaisi käyttöjännite. FLSender-sovelluksen käynnistyessä luetaan aina konfigurointitiedosto FLASH-levyltä ja annetaan se Settings-oliolle parsittavaksi metodilla setProperties. Tämän jälkeen asetukset ovat sovelluksen käytettävissä.

5.2.11 Sovelluksen testaus

FLSender-sovellukselle on laadittu testisuunnitelma perustuen mustalaatikkotestaukseen. Mustalaatikkotestaus perustuu toiminnallisiin ja laadullisiin vaatimuksiin. Mustalaatikkotestauksessa ei välitetä testattavan kohteen rakenteesta tai sisällöstä, vaan tutkittavana ovat kohteen tulosteet erilaisilla syötearvoilla. Testaajalle kohde on tuntematon "musta laatikko". Testattavan kohteen oikeellisuutta tarkastellaan vertaamalla saatuja tulosteita haluttuihin tai odotettuihin tulosteisiin. Testitapaukset johdetaan aina kohteen määrittelyn perusteella. Testauksessa käytetään TestLog-nimistä kaupallista sovellusta, jolla

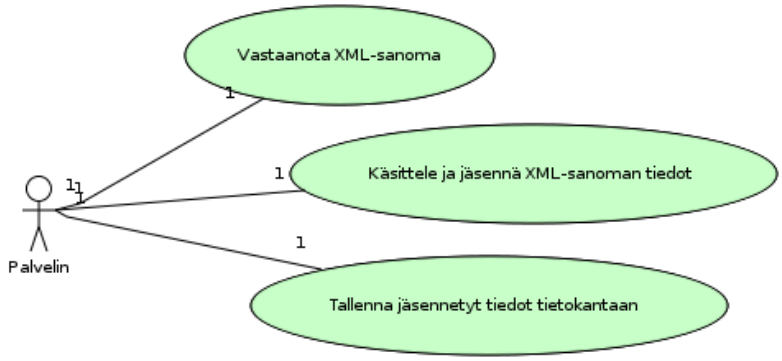
on laadittu testitapaukset FLsender-sovellukselle. TestLogilla on helppo ylläpitää testitapauksia ja lisäksi se osaa tuottaa erilaisia raportteja testiaineistosta.

5.3 Palvelinohjelmisto - FLReceiver

FLReceiver on JavaServlet-tekniikalla ohjelmoitu komponentti ajoneuvotietokoneen lähettämien XML-viestien vastaanottamiseen ja käsittelemiseen. FLReceiver käyttää tietokantayhteyksien allasta (pool) monen yhtäaikaisen tietokantayhteyden ylläpitämiseen. Tämä tekee tietojen kirjoittamisesta tietokantaan nopeampaa kuin, että tarvittavat yhteydet avattaisiin uudestaan jokaista palvelutoimepidettä varten. Tietokantayhteyksien varastoiminen altaaseen toteutetaan helppokäyttöisellä avoimen lähdekoodin Proxool-nimisellä kirjastolla.

5.3.1 Sovelluksen suunnittelunäkökulmat

Kuvassa 34 oleva FLReceiver-sovelluksen käyttötapauskaavio sisältää kolme erilaista käyttötapausta. Palvelimella sijaitsevan sovelluksen tehtävänä on vastaanottaa ajoneuvotietokoneen lähettämä XML-sanoma, jäsentää sanoman tiedot tietovarastopaukkauksen (fi.sesca.a1.storage) olioihin ja tallentaa oliorakenteen tiedot pysyvästi tietokantaan.

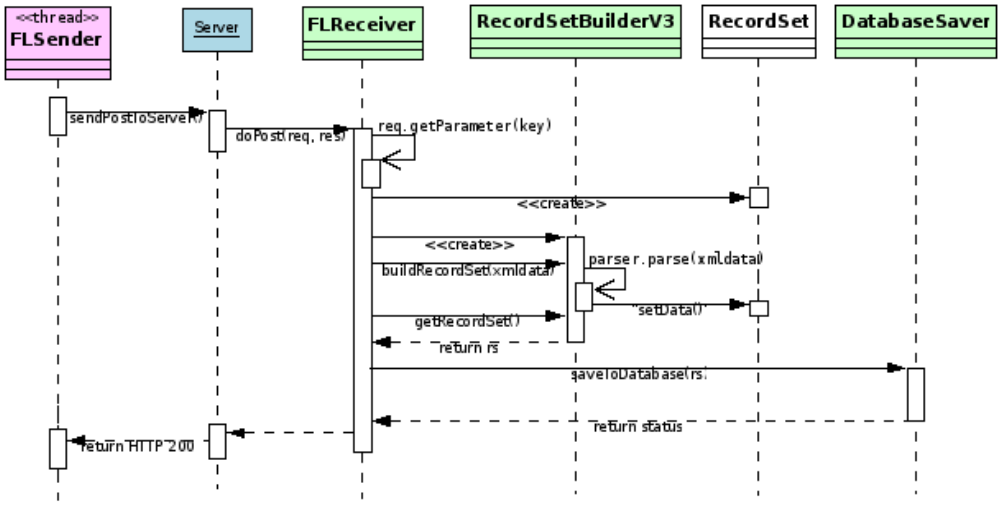


Kuva 34. FLReceiver-sovelluksen käyttötapauskaavio

FLReceiver-sovellus koostuu kahdesta pakkauksesta fi.sesca.a1.receiver ja fi.sesca.a1.storage. Pakkaus fi.sesca.a1.receiver sisältää ohjelmiston toiminnan toteuttavat luokat ja pakkaus fi.sesca.a1.storage on oliotietorakenteen toteuttava luokka, johon XML-viestin sisältämät tiedot puretaan väliaikaisesti ennen tietokantaan tallennusta. FLSender- ja FLReceiver-sovelluksille on ohjelmoitu tarkoituksella yhteinen tietovarastopakkaus fi.sesca.a1.storage, joka nopeuttaa omalta osaltaan ohjelmistoon tehtäviä muutoksia. Siten tietorakenteeseen tehty uusi ominaisuus tai muutos on automaattisesti käytettävissä kummastakin ohjelmasta.

5.3.2 *Tietojen vastaanottaminen*

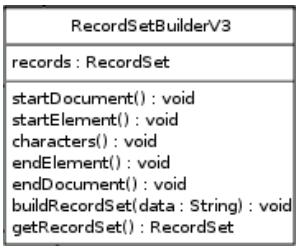
Tärkein fi.sesca.a1.receiver-pakkauksen luokka on FLReceiver, joka perii HTTP-servlet luokan ominaisuudet. FLReceiver-luokka hallitsee sovelluksen kokonaisuutta eli käsittelee asiakkaan (ajoneuvo-ohjelmiston) lähettämän HTTP POST -kyselyn, siirtää XML-sanoman tiedot oliotietorakenteeseen ja tallentaa sen sisältämät tiedot tietokantaan. Kuvassa 35 ajoneuvotietokoneen ohjelmisto (luokka FLSender) kutsuu FLReceiver-luokkaa palvelinohjelmiston välityksellä lähettäessään XML-viestin. FLReceiver-luokan tärkein metodi on doPost, jota palvelimen servlet-säiliö kutsuu automaattisesti FLSenderin tekemän HTTP POST -kyselyn seurauksena. Metodissa doPost luetaan ensiksi HTTP-kyselyn sisältämä avain. Avaimen selvittämisen jälkeen FLReceiver lukee kyselystä XML-viestin ja luo viestin sisältämiä tietoja varten RecordSet-tietovarasto-olion. Tämän jälkeen se luo aiemmin lukemaansa avaimeen sopivan XML-viestin käsittelijäolion RecordSetBuilderV3 ja pyytää käsittelijää parsimaan XML-viestin. Kun käsittelijä on tehnyt työnsä FLReceiver pyytää siltä XML-viestin sisältämät tiedot pakattuna RecordSet-tietovarasto-olioon metodilla getRecordSet. Lopuksi FLReceiver pyytää DatabaseSaver-oliota viemään RecordSet-olion sisältämät tiedot tietokantaan. DatabaseSaver palauttaa statuksen onnistumisesta. Palvelin-ohjelmisto palauttaa viimeisenä operaationaan HTTP 200 -statuskoodin FLSender-sovellukselle HTTP-kyselyn onnistumisesta.



Kuva 35. FLReceiver-luokka hallitsee tietojen vastaanottamista ja tallentamista

Kuvan 36 FLReceiver-luokka koostuu DatabaseSaver- ja FileSaver -apuluokista. DatabaseSaver-luokan tehtävä on tallentaa sille välitetty pakkauksen fi.sesca.a1.storage mukainen oliotietorakenne tietokantaan ja FileSaver-luokan tehtävä on tallentamaa XML-sanoma, jota ei pystytä käsittelemään palvelimen lokihakemistoon.

Kuva 36. FLReceiver-pääluokka



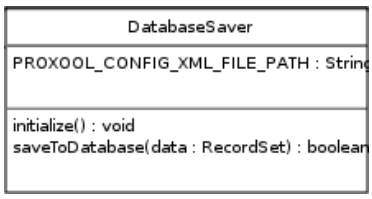
Kuva 37. Osana Builder-suunnittelumallia toteutettu konkreettinen rakentajaluokka

FLReceiver-sovelluksessa XML-viestit käsitellään käyttäen SAX-parseria. SAX-parseri määrittelee rajapinnan, jolla mikä tahansa kieliopiltaan oikea XML-viesti pystytään lukemaan. Ohjelmoijan tehtäväksi jää ohjelmoida rajapinnan toteuttavat konkreettiset luokat. Kuvassa 37 on yksi konkreettinen XML-viestin käsittelijäluokka RecordSetBuilderV3. Tarkemmin se on osa rakentaja-suunnittelumallia (builder), josta SAX-parseri toteuttaa suurimman osan. SAX-parserin käyttäjän tulee ohjelmoida vain konkreettiset toteutukset eli XML-viestin käsittelijät. Nämä käsittelijät ovat rakentaja-suunnittelumallin builder-luokkia. Liitteestä 3 nähdään, että FLReceiver-sovelluksessa konkreettisia builder-luokkia (XML-käsittelijöitä) on kaiken kaikkiaan 3 erilaista - kullekin FLSenderin lähettämälle HTTP POST -kyselyn avaimelle omansa.

5.3.3 Tietojen tallentaminen

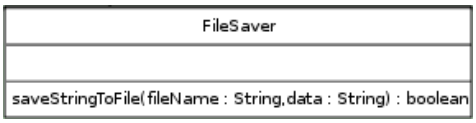
Kuvassa 38 oleva DatabaseSaver-luokka vastaa tietojen tallentamisesta tietokantaan. Luokan initialize-metodissa alustetaan tietokantayhteyksien allas. Tietokantayhteysallas tarkoittaa palvelimen muistiin valmiiksi luotuja tietokantayhteysolioita, joita jaetaan sujuvasti yhteyksien käyttäjille. FLReceiver-sovellus sijaitsee JavaServlet-säiliössä, ja jokainen siihen kohdistunut palvelupyyntö ajetaan Apache Tomcat -sovelluspalvelimella sisäisesti omana säikeenä. Koska DatabaseSaver-luokka on FLReceiver-luokan jäsenmuuttujana jokainen servlet-palvelimella luotu ja ajettava FLReceiver-säie jakaa DatabaseSaver-luokan ja siten sovellukselle yhteisen tietokantayhteyksien altaan. Tietokantayhteysallas on toteutettu avoimen lähdekoodin Proxool-nimisellä Java-kirjastolla. Helppokäyttöisenä kirjastona Proxool käärii pyydettyjen tietokantayhteyksien käytön. Sovellusohjelmoija pyytää kirjastolta tietokan-

tayhteyttä, joka on sovelluksen käytössä aina yhteyden sulkemiseen asti. Proxool-kirjastolle konfiguroidaan maksimitietokantayhteyksien lukumäärä, jonka se enintään saa avata.



Kuva 38. Luokka DatabaseSaver tallentaa tiedot pysyvästi tietokantaan

Jos yhteyksiä on enemmän, mitä maksimikonfiguraatio määrittää FLReceiver-servletsäikeet jäävät odottamaan vapautuvia yhteyksiä. Tietokantayhteyksien altaan käytöstä on ainakin kahdenlaista hyötyä. Hyötyä saadaan siinä, että yhteydet avataan vain kerran, ja ne ovat avaamisen jälkeen aina käytössä. Siten luotu yhteys voidaan saman tien antaa uusien FLReceiver-servlet säikeiden käyttöön ilman, että tietokantayhteys tarviisi luoda joka kerta uudestaan säiekohtaisesti. Tämä lisää järjestelmän suorituskykyä, koska asiakkaan tekemiin kyselyihin pystytään vastata nopeammin. Toinen hyöty saadaan palvelimen kuormituksen hallinnassa. Tietokantayhteysallas voidaan mitoittaa siten, että se pystyy palvelemaan tehokkaasti monia asiakkaita, mutta toisaalta niin, että se ei anna ylikuormittaa palvelinta. Mikäli jokaiselle asiakkaalle luotaisiin oma tietokantayhteys, se pahimmassa tapauksessa, asiakkaiden lukumäärän ollessa hyvin suuri, aiheuttaisi palvelimen ylikuormittumisen ja siten palvelimen jumiutumisen.



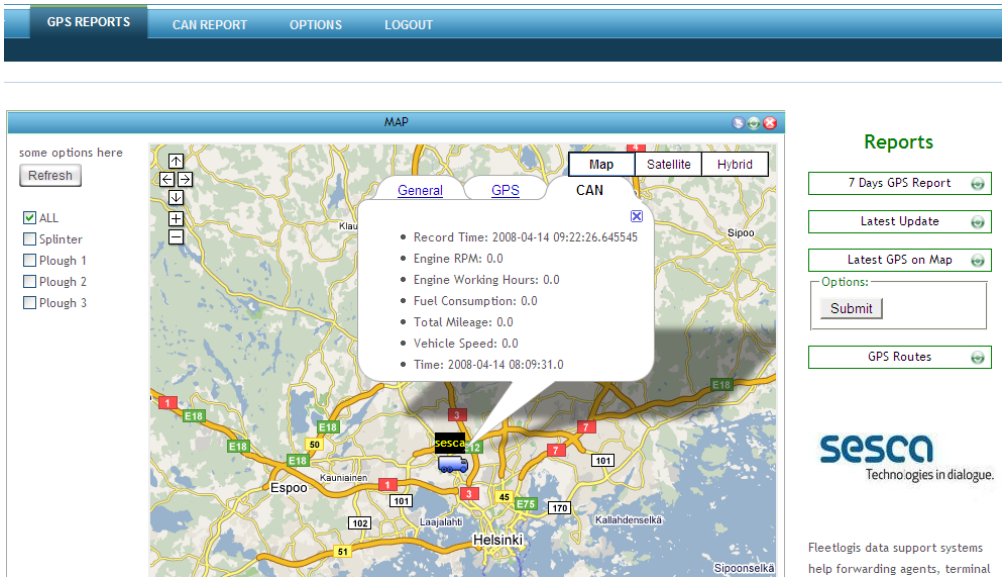
Kuva 39. Tiedostojentallennusluokka

Kuvassa 39 olevan FileSaver-luokan tehtävänä on tallentaa vastaanotettu XML-viesti palvelimen lokihakemistoon, mikäli viestiä ei pystytä käsittelemään. Luokalla on ainoastaan yksi metodi, jonka ensimmäinen parametri määrittää polun tallennettavalle XML-viestille ja toisena parametrina metodille annetaan tallennettava XML-viesti.

5.4 Palvelinohjelmisto - FLReports

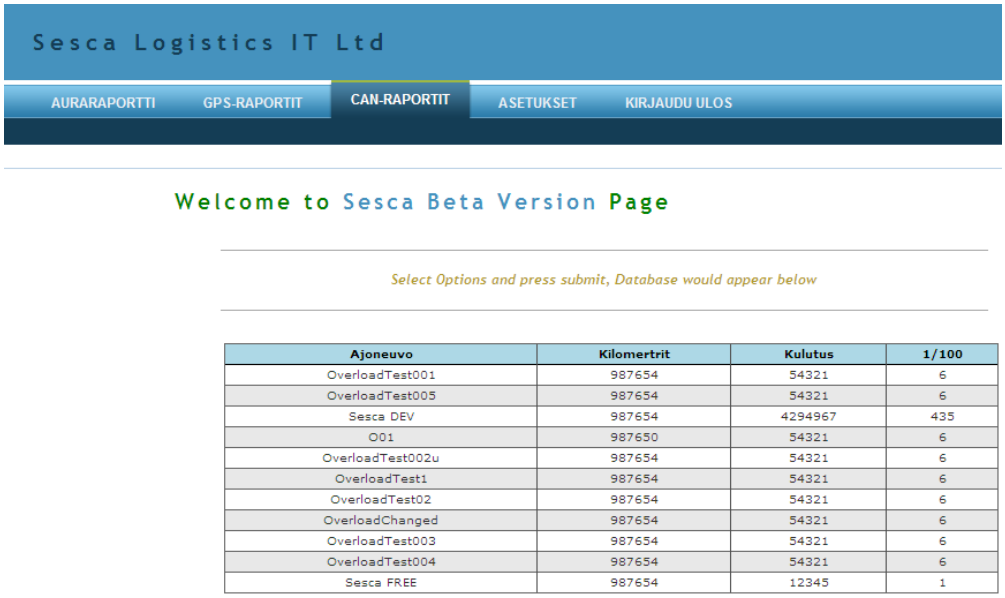
FLReports-komponenttia ei ole toteutettu tässä työssä vaan sen on toteuttanut Sesca Logistics IT Oy:n palveluksessa työskentelvä Mohammad Saffaf. FLReports liittyy oleellisesti insinöörityöhön tarjoamalla helpon ajoneuvojen tietojen hakemisen ja selaamisen WWW-pohjaisella käyttöliittymällä. Käyttöliittymä on lokalisoitu suomen-, englannin- ja arabiankieliseksi. FLReports-komponentti sisältää kolme erilaista GPS-raporttia, kolme erilaista CAN-raporttia, auraraportin sekä suolasirotinraportin. GPS-raportit on suunniteltu erityisesti näyttämään viimeisimpiä ajoneuvojen sijainteja kartalla. CAN-raporteilla on mahdollista hakea ajoneuvojen moottoreista kerättyjä kumulatiivisia arvoja, valitun aikavälin laskennalliset arvot tai CAN-raakatietoja valitusta ajoneuvosta.

Kuvan 40 GPS-raportista nähdään ajoneuvojen viimeisin sijaintitieto kartan päällä. Ajoneuvot on piirretty kartalle kuorma-auton kuvina ja laitettaessa kohdistin ajoneuvon päälle saadaan esiin kyseisen ajoneuvon tarkemmat tiedot. Yleisten tietojen välilehdellä (general) listataan ajoneuvon nimi ja mittaustapahtuman aikatieto. GPS-välilehdellä ovat viimeisimmät paikkatiedot tekstimuodossa leveys- ja pituusasteina. CAN-välilehdellä ovat kuvassa 40 näkyvät ajoneuvon nopeus, ajetut kilometrit, kulutettu polttoainemäärä, moottorin kierroslukunopeus ja moottorin käyntitunnit. Raportti on tarkoitettu ajoneuvojen reaaliaikaiseen seurantaan. Raportin karttaominaisuus on toteutettu GoogleMap-karttapalvelulla. GoogleMap-karttapalvelu tarjoaa tavallisen karttapohjan lisäksi satellittikuvat sekä kartan ja satellittikuvien yhdistetyn näkymän.



Kuva 40. GPS-raportti (Latest GPS on Map) näyttää ajoneuvojen sijainnin kartan päällä

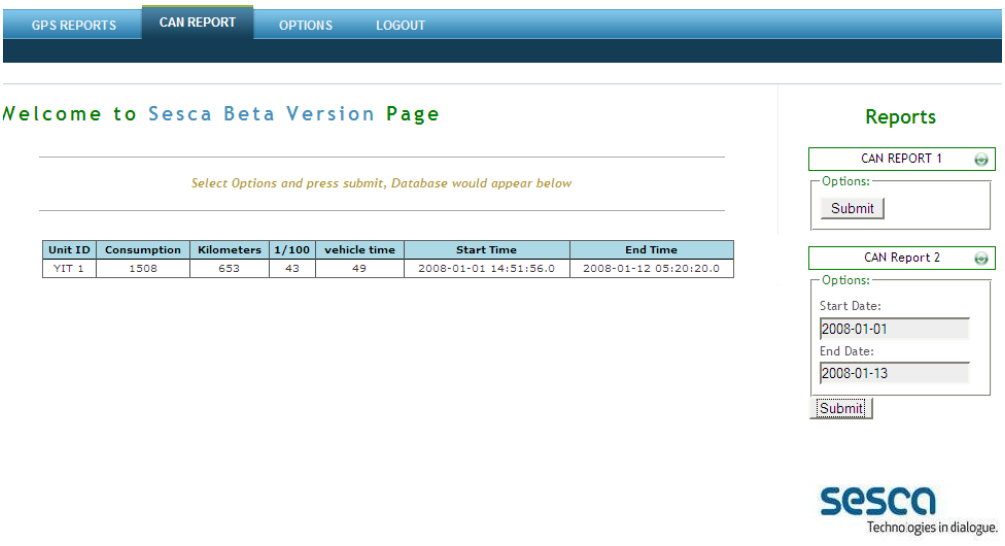
Kuvassa 41 olevalla CAN-raportilla sadaan kaikkien ajoneuvojen ajama kumulatiivinen kilometrimäärä sekä ajoneuvon kuluttama kumulatiivinen polttoainemäärä litroina. Raporttiin on myös laskettu ajoneuvon keskikulutus sataa kilometriä kohti perustuen kumulatiiviseen kilometrimäärään ja polttoainekulutukseen.



Ajoneuvo	Kilometrit	Kulutus	1/100
OverloadTest001	987654	54321	6
OverloadTest005	987654	54321	6
Sesca DEV	987654	4294967	435
O01	987650	54321	6
OverloadTest002u	987654	54321	6
OverloadTest1	987654	54321	6
OverloadTest02	987654	54321	6
OverloadChanged	987654	54321	6
OverloadTest003	987654	54321	6
OverloadTest004	987654	54321	6
Sesca FREE	987654	12345	1

Kuva 41. CAN-raportti, joka listaa ajoneuvojen ajamat kilometrit sekä kumulatiivisen kulutuksen

Kuvassa 42 on CAN-raportti, jolla haetaan ajoneuvojen tiedot tietyltä aikaväliltä. Raportissa esitetään aikavälin laskennalliset arvot. Esimerkiksi kuvan 42 tienhoitoajoneuvo on ajanut 1.1.2008 - 13.1.2008 välisellä ajanjaksolla 683 kilometriä, kuluttanut 1508 litraa polttoainetta ja ajoneuvon moottori on ollut käynnissä 49 tuntia. Ajoneuvon keskikulutus on ollut 43 litraa sadalla kilometrillä.



Unit ID	Consumption	Kilometers	1/100	vehicle time	Start Time	End Time
YIT 1	1508	653	43	49	2008-01-01 14:51:56.0	2008-01-12 05:20:20.0

Kuva 42. CAN-raportti, jolla haetaan ajoneuvojen tietoja valitulta ajanjaksolta

6 TULOKSET

Fleetlogis-tietojärjestelmän ajoneuvotietokoneen FLSender-sovellusta voidaan pitää edistyneenä ratkaisuna. Se hyödyntää Javaa sulautettujen järjestelmien toteutuskielenä, mikä puolestaan mahdollistaa nopean ja joustavan ajoneuvotietokoneen ohjelmiston kehittämisen ja ylläpidon. Aiempaa oliotietämystä voi suoraan hyödyntää sovelluksen kehittämisessä.

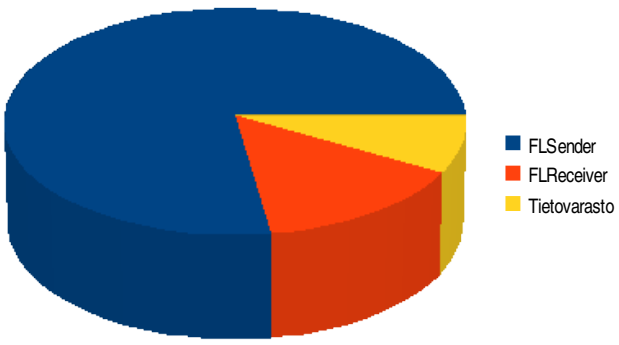
Ajoneuvotietokoneen FLSender-sovelluksen kehittämisessä panostettiin suunnitteluun. Hyvän suunnittelun ansiosta ohjelmisto tarjoaa esimerkiksi selkeän mallin, kuinka liittää uusia tienhoitoajoneuvon laitteita sovellukseen. Tämä on merkittävä hyöty sovelluksen jatkokehityksen kannalta. Sovelluksen seuraaviin versioihin onkin kaavailtu kahden muun suolasirotinvalmistajan laitteiden liittämien. Yleisesti ottaen onnistuttiin luomaan sovellus, jota voidaan joustavasti jatkokehittää kunkin asiakkaan tarpeiden mukaan.

Taulukkoon 3 on listattu insinöörityössä syntyneet kolme komponenttia ja niiden koot. Ajoneuvotietokoneen ohjelmisto sisältää n. 6000 koodiriviä ja palvelinpään tietojen vastaanottosovellus noin 1100 koodiriviä. Kummallakin sovelluksella on yhteinen tietovarastokomponentti, jonka koko on n. 600 koodiriviä. Sovellusten yhteenlaskettu koko on 7696 koodiriviä.

Taulukko 3. Insinöörityössä toteutettujen komponenttien koot

Sovellus	**Kuvaus**	**Koko (koodiriviä)**	**Suhteellinen koko (%)**
FLSender	Ajoneuvotietokoneen sulautettu Java-ohjelmisto	5953	77,4
FLReceiver	Palvelimelle oleva tietojen vastaanottosovellus	1117	14,5
Tietovarasto	FLSender- ja FLReceiver-sovellusten käyttämä tietovarastokomponentti	626	8,1

Kuva 43 esittää insinöörityössä syntyneiden sovellusten suhteellista kokoa. Ajoneuvotietokoneen FLSender-sovellus on ylivoimaisesti suurin kolmesta komponentista.



Kuva 43. FLSender-, FLReceiver- ja tietovarastokomponentin suhteellinen koko

7 YHTEENVETO

Insinöörityössä suunniteltiin ja toteutettiin raskaassa kalustossa olevaan ajoneuvotietokoneeseen ohjelmisto, joka kerää tietoja ajoneuvosta ja ajoneuvoon liitetyistä erilaisista lisälaitteista. Insinöörityössä toteutettiin myös palvelinpään komponentti, joka ottaa vastaan ajoneuvotietokoneen ohjelmiston lähettämät XML-sanomat ja tallentaa sanoman tiedot pysyvästi tietokantaan. Työn tuloksena toteutettu Fleetlogis-tietojärjestelmä sopii ensisijaisesti tienhoito- ja maansiirtologistiikanalan yritysten tarpeisiin. Sitä on myös suhteellisen helppo laajentaa samankaltaisten muiden alojen käyttöön. Kansanvälisenä tietojärjestelmänä Fleetlogisilla on myös mahdollisuudet saavuttaa ulkomaalaiset asiakkaat.

VAMOS-hankkeen kehitysprojektissa oli paljon tavoitteita. Osa tavoitteista oli yhteisiä kaikille asiakkaille ja osa asiakaskohtaisia. Monia tavoitteita saavutettiin ja erityisesti asiakkaiden yhteisiin tavoitteisiin onnistuttiin vastaaman hyvin. Esimerkiksi ajoneuvojen sijaintitiedot ovat saatavissa informatiivisina karttaraportteina. Tämä lisää mm. työturvallisuutta, koska yritysten tarve kommunikoida suoraan ajoneuvoon matkapuhelimella vähenee. Myös ajoneuvon CAN-väylältä saadaan siirrettyä matka-, kulutus-, moottorin nopeus- ja käyntiaika- sekä ajoneuvon nopeustieto palvelimelle, josta ne ovat hyödynnettävissä erilaisin raportein. Tämän ansiosta yrityksillä on aiempaa paremmat mahdollisuudet seurata kustannustehokkuuttaan. Kaikkien asiakkaiden yhteisistä tavoitteista jäi puuttumaan digipiirturilta saatavat tiedot kuten kuljettajatunnistus ja kuljettajakohtaiset työaikakirjaukset. Tienhoitoalan asiakaskohtaisissa tavoittessa onnistuttiin kohtalaisesti. Etuauran ja alusterän käyttöä pystytään seuraamaan palvelimella olevan raportin avulla. Sivuauran toteuttamisessa kohdattiin teknisiä ongelmia auran signaalin tulkitsemisessa ja sen toteutusta päätettiin lykätä. NIDO-tyyppisen suolasirotinlaitteen toteutus on kehitysvaiheessa. Ajoneuvojen valmistajan haasteet tulivat kehityshankkeessa ilmi, mutta niihin ei päätetty varsinaisesti tässä vaiheessa lähteä vastaamaan.

Kehitetystä Fleetlogis-tietojärjestelmästä julkaistiin asiakkaalle 10.3.2008 beetaversio (Beta Release 1), joka sisälsi yksinkertaisia raportteja ajoneuvotietokoneen tietojen selaamiseen. Ensimmäinen varsinainen tuotejulkaisu (Release Candidate 1) on suunniteltu tehtäväksi 15.5.2008. Tällöin asiakkaat saavat monipuolisemman ja koodivalmiin tietojärjestelmän, jolla voi seurata reaaliaikaisesti ajoneuvojen sijainteja kartalla sekä ajoneuvoista kerättäviä muita tietoja. VAMOS-hanke päättyy 31.5.2008. Kaikkia järjestelmälle asetettuja tavoitteita ei kyetty toteuttamaan, ja siksi järjestelmää olisi luonnollista lähteä jatkokehittämään uudella projektilla.

VIITELUETTELO

[1] Tekes. *VAMOS - Liiketoiminnan mobiilit ratkaisut 2005-2010* [verkkodokumentti]. [viitattu 27.01.2008]. Saatavissa http://www.tekes.fi/ohjelmat/vamos/ .

[2] Toivonen, Mikko, VAMOS hankesuunnitelma [rajoitettu Word-dokumentti] [viitattu 07.03.2008].

[3] Peltomäki, Juha, *J2ME-ohjelmointi*, 2004, 1. painos.

[4] Muchow, John W., *CORE J2ME Technology & MIPD*, 2002.

[5] Toply, Kim, *J2ME IN A NUTSHELL A Desktop Quick Guide*, 2002, 1. painos

[6] Wikipedia, Java Platform, Micro Edition [verkkosivu], [viitattu 10.02.2008]. Saatavissa: http://en.wikipedia.org/wiki/J2ME .

[7] Mahmoud, Qusay H, Getting Started With the Information Module Profile, March 2006, [verkkosivu] [viitattu 03.04.2008]. Saatavissa: http://developers.sun.com/mobility/imp/impstart/ .

[8] Gamma, Helm, Johnson, Vlissides, *Design Patterns*, 2001.

[9] Aplicom Oy, APLICOM A1 VEHICLE TELEMATICS UNIT W/JAVA A1 FLEX Product description [rajoitettu PDF-dokumentti] [viitattu 08.03.2008].

[10] RS-485, Wikipedia [verkkosivu] [viitattu 08.03.2008]. Saatavissa: http://fi.wikipedia.org/wiki/RS-485 .

[11] RS-232, Wikipedia [verkkosivu] [viitattu 08.03.2008]. Saatavissa: http://fi.wikipedia.org/wiki/RS-232 .

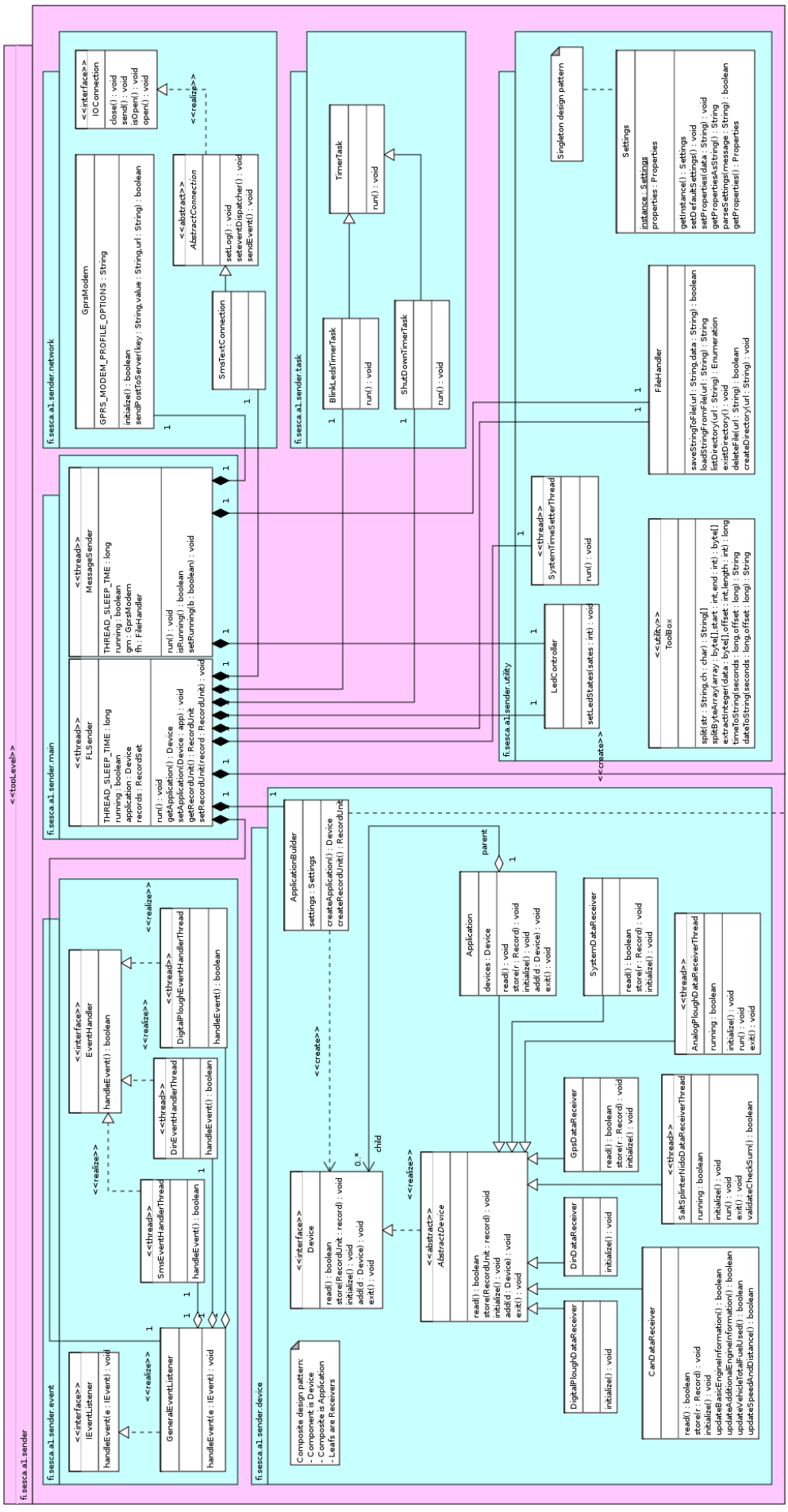
[12] Aplicom A1 Software Developer's Guide rev. 3.2 [rajoitettu PDF-dokumentti] [viitattu 08.03.2008].

[13] Siemens, TC65 JAVA User's Guide [PDF-dokumentti] [viitattu 08.03.2008]. Saatavissa: http://www.tc65.ru/lib/tc65_java_user_guide.pdf .

[14] Aplicom Oy, A1 PROGRAMMER'S CHECKLIST [rajoitettu PDF-dokumentti] [viitattu 08.03.2008].

[15] Unified Modeling Language, Wikipedia [verkkosivu] [viitattu 13.03.2008]. Saatavissa: http://en.wikipedia.org/wiki/Unified_Modeling_Language .

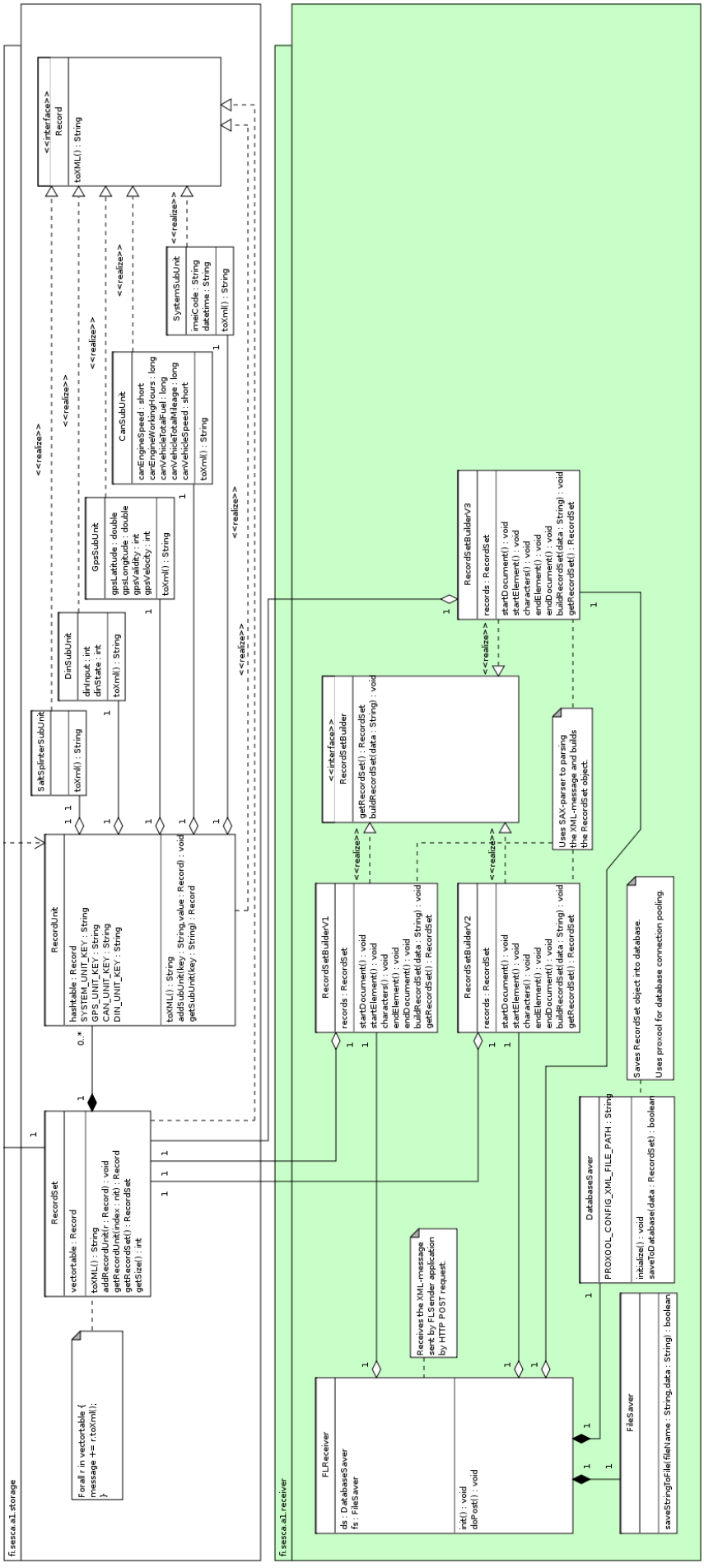
FLSENDER-SOVELLUKSEN LUOKKAKAAVIO



FLSENDER-SOVELLUKSEN LÄHETTÄMÄ XML-VIESTI

```
<?xml version="1.0" encoding="UTF-8"?>
<message>
     <record>
       <device>
           <imei>000000000000006</imei>
           <datetime>2008-03-07 10:00:00</datetime>
       </device>
       <gps>
           <latitude>60.2218</latitude>
           <longitude>24.8778</longitude>
           <validity>2</validity>
           <velocity>60.4</velocity>
       </gps>
       <can>
           <engine_speed>800</engine_speed>
           <engine_working_hours>987654321</engine_working_hours>
           <vehicle_fuel>54321</vehicle_fuel>
           <vehicle_mileage>987650000</vehicle_mileage>
           <vehicle_speed>909</vehicle_speed>
       </can>
       <din>
           <input>1</input>
           <state>1</state>
       </din>
     </record>
     <record>
       <device>
           <imei>000000000000006</imei>
           <datetime>2008-03-07 10:00:00</datetime>
       </device>
       <gps>
           <latitude>60.2218</latitude>
           <longitude>24.8778</longitude>
           <validity>2</validity>
           <velocity>82.5</velocity>
       </gps>
       <can>
           <engine_speed>7851</engine_speed>
           <engine_working_hours>555</engine_working_hours>
           <vehicle_fuel>16534</vehicle_fuel>
           <vehicle_mileage>987654321</vehicle_mileage>
           <vehicle_speed>810</vehicle_speed>
       </can>
       <din>
           <input>1</input>
           <state>0</state>
       </din>
     </record>
</message>
```

FLRECEIVER-SOVELLUKSEN LUOKKAKAAVIO



FLSENDER-SOVELLUKSEN KONFIGUROINTITIEDOSTO

```
flsender_version: 0.7
use_power_suspend: true
waiting_time_before_use_power_suspend: 900000
program_sleep_time: 300000
message_sender_sleep_time: 600000
time_offset: 7200000
fl_receiver_url: http://www.fleetlogis.com:8080/FLServer/FLReceiver
gprs_bearer_type: gprs
gprs_access_point: internet
com2_debug: true
com2_info: true
system_logging: true
gps_logging: true
can_logging: true
splinter_logging: true
plough1_logging: false
plough2_logging: false
plough3_logging: false
plough1_high_din_channel: 1
plough1_low_din_channel: 2
plough2_high_din_channel: 5
plough2_low_din_channel: 6
plough3_analog_io_channel: 3
plough1_event_after_cum_up_time: 3000
plough2_event_after_cum_up_time: 1750
plough3_thread_sleep_time_after_poll: 500
plough3_state_threshold_value: 1500
din_logging: false
din1_logging: false
din2_logging: false
din3_logging: false
din4_logging: false
din5_logging: false
din6_logging: false
ad_din_ref_level: 4
din_hysteresis: 1
salt_splinter_type: 1
com1_baud_rate: 9600
com1_bitsperhchar: 8
com1_stopbits: 1
com1_parity: none
com1_blocking: on
com1_autocts: on
com1_autorts: on
```